(4)

Learning in Structured Connectionist Networks

Mark Alan Fanty

Technical Report 252
April 1988

UNIVERSITY OF
ROCHESTER
COMPUTER SCIENCE

Learning in Structured Connectionist Networks

Mark Alan Fanty
Computer Science Department
University of Rochester

April 1988

TR 252

Submitted in Partial Fulfillment of the
Requirements for the Degree
Doctor of Philosophy

Supervised by Jerome Feldman
Computer Science Department
University of Rochester
Rochester, New York

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: TR252	2. GOVT ACCESSION NO.
3. RECIPIENT'S CATALOG NUMBER	
4. TITLE *(and Subtitle)*: Learning in Structured Connectionist Networks	5. TYPE OF REPORT & PERIOD COVERED: Technical Report
	6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s): Mark Alan Fanty	8. CONTRACT OR GRANT NUMBER(s): N00014-04-K-0655
9. PERFORMING ORGANIZATION NAME AND ADDRESS: Dept of Computer Science 734 Computer Studies Bldg. Univ. of Rochester, Rochester, NY 14627	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS: DARPA / 1400 Wilson Blvd. Arlington, VA 22217	12. REPORT DATE: April 1988
	13. NUMBER OF PAGES: 89
14. MONITORING AGENCY NAME & ADDRESS *(if different from Controlling Office)*: Office of Naval Research Information Systems Arlington, VA 22217	15. SECURITY CLASS. *(of this report)*: unclassified
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT *(of this Report)*
Distribution of this document is unlimited.

17. DISTRIBUTION STATEMENT *(of the abstract entered in Block 20, if different from Report)*

18. SUPPLEMENTARY NOTES

19. KEY WORDS *(Continue on reverse side if necessary and identify by block number)*
Learning, Connectionist, Massively Parallel

20. ABSTRACT *(Continue on reverse side if necessary and identify by block number)*

-- Connectionist networks compute in a manner analogous to real neural networks. The work in this thesis focuses on computing and especially learning in structured connectionist networks-those which emphasize problem-specific connection patterns as well as adaptive weight change rules. A connectionist chart parser was implemented which parses limited-length strings for context-free grammers in a constant number of parallel computation steps. The parser was extended to disambiguate and complete near-miss parses, as well as learn new productions in certain limited situations. While the parser works well,

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

20. ABSTRACT (Continued)

the structure is too rigid and learning too difficult for cognitive modeling. Two algorithms for learning simple, feature-based concept descriptions were also implemented. The first recruits hidden units representing pairs of features. The performance of this network is good when the definitions involve pairs of input features, but attempts to build hierarchies of pair units for longer definitions were not successful. The second algorithm is an enhancement of an existing technique, competitive learning, adding feedback from concept units to guide the partitioning of inputs into classes. This technique is more successful and is used as a component in a network which is capable of learning descriptions of structured objects. A key assumption for this work is the need to process the primitives of a structured object sequentially in order to avoid cross talk. Implementing this sequential processing within the connectionist paradigm presents various difficulties, especially in the context of learning. The implementation is preliminary, but promising.

Acknowledgements

This research was supported by the Office of Naval Research, contract number N00014-04-K-0655. The International Computer Science Institute supported me during the final stages of the work. I would like to thank my wife Martha for enduring so many years of impoverished graduate student life and for all the support she provided during those years. She also proofread this document, finding many typos and twisted sentences. No class ever taught me as much as the students and staff at Rochester, who were always willing to help. I am especially indebted to Liudvikas Bukys and Stuart Friedberg who rescued my code on countless occasions. I am grateful for the attention and advice of my thesis committee, Prof. James Allen, Prof. Dana Ballard and Prof. Gary Dell. Finally, I would like to thank my advisor, Prof. Jerome Feldman, who taught me about connectionist computation and helped me in many ways. Without his extra efforts, this thesis would not have happened.

Abstract

Connectionist networks compute in a manner analogous to real neural networks. The work in this thesis focuses on computing and especially learning in structured connectionist networks—those which emphasize problem-specific connection patterns as well as adaptive weight change rules. A connectionist chart parser was implemented which parses limited-length strings for context-free grammers in a constant number of parallel computation steps. The parser was extended to disambiguate and complete near-miss parses, as well as learn new productions in certain limited situations. While the parser works well, the structure is too rigid and learning too difficult for cognitive modeling. Two algorithms for learning simple, feature-based concept descriptions were also implemented. The first recruits hidden units representing pairs of features. The performance of this network is good when the definitions involve pairs of input features, but attempts to build hierarchies of pair units for longer definitions were not successful. The second algorithm is an enhancement of an existing technique, competitive learning, adding feedback from concept units to guide the partitioning of inputs into classes. This technique is more successful and is used as a component in a network which is capable of learning descriptions of structured objects. A key assumption for this work is the need to process the primitives of a structured object sequentially in order to avoid cross talk. Implementing this sequential processing within the connectionist paradigm presents various difficulties, especially in the context of learning. The implementation is preliminary, but promising.

Contents

List of Tables

List of Figures

Chapter 1

INTRODUCTION

The research presented in this thesis is connectionist in a strong way; it is a priori connectionist. The guiding assumption is that connectionist models will outperform traditional techniques on a variety of tasks, especially those which have resisted formulation in logic. The work described below represents several efforts towards establishing some fundamental techniques in connectionist computation—techniques upon which future, more sophisticated work can build. The title of the thesis reflects my bias. I believe that learning is central to the success of connectionist models. Adaptability is, in fact, one of the major reasons for using connectionist networks. This bias is shared by most connectionist researchers. I also believe that learning algorithms cannot do everything. If we want networks to solve complex problems, we need to build them in a manner suited to the problem; we need to build in structure.

There are two distinct lines of work reported here. The first is a connectionist parser. This is a highly structured network, with easily analyzed behavior. While some learning results are given, learning is not emphasized. The second line of work involves concept learning. Two different learning techniques are given for simple, feature-based concepts—those with no structure. One of these is extended to work with structured objects, a much more difficult task for connectionist networks. The use of continual, automatic shifts of attention is used to process the structured objects while avoiding crosstalk. The use of sequential processing at this level in connectionist networks is an important and controversial assumption. Its utility is supported by a sketch of an inferencing mechanism which uses attention to do binding. More specifically, the contributions of this thesis are:

- Fast, exact implementation of bottom-up context-free parsing in a connectionist network. The major limitations are the number of units needed and the rigidity of the parser. The maximum length of acceptable input is strictly limited.
- Extension of the parser to disambiguate and implementation of "near-miss" parsing, which will complete parses missing some productions with decreasing activation for incomplete parses. The notion of near-miss is limited, e.g. missing words cannot be handled.
- Development of learning techniques for the parser which are able to distribute a rule learned locally throughout the network using a new technique. Global rule

units detect the local learning. During idle periods, they create local conditions throughout the network which institute the same learning everywhere.

- Development of a hidden-unit recruitment rule for supervised learning tasks which uses minimal feedback to form representations of important pairs of inputs. The learning is fast and works well when only pairs are needed. However, the extension to longer conjunctions of inputs proved unsuccessful.

- Augmentation of competitive learning to work in a supervised learning situation. Top-down feedback from the output units biases the competitive learning units to partition the input according to the reinforcement given.

- Implementation of a structured-object learning network. The key assumption is that primitive components of a structured object must be processed sequentially. This necessitates development of a mechanism to automatically switch attention between the primitives and requires the use of learning techniques which can operate in a dynamic environment where the input seen by the hidden units continually changes during one session. The implementation is preliminary; however, many interesting issues are addressed.

1.1 Overview of Connectionist Computation

Several introductions to connectionist models can be found in the literature [Feldman and Ballard, 1983, Rumelhart and McClelland, 1986, Waltz and Feldman, 1987, Feldman *et al.*, 1988]. This section describes the particular style of connectionist computing used throughout this thesis. Connectionist units (hereafter "units") are patterned after neurons, but they are not exact neural models. The units are simple computational devices with very limited memory. The are connected by links. Each unit computes a single output value, which is sent down all links emanating from that unit and used by the receiving units to compute their next output.

Compared to much of the literature, the units used in this thesis are complex and heterogeneous. Figure 1 illustrates a typical unit. Each unit has a numerical *potential*, or level of activation. This activation should not encode complex or symbolic information. For example, if a unit represents the hypothesis that a certain word is a noun, then the potential of that unit might encode the strength of belief in that hypothesis. It would be inappropriate to have a unit with potential 0.1 for noun, 0.2 for verb, etc. Each unit also has an *output* value which, in all the networks presented in this thesis, is a simple, monotonic function of the potential. Intuitively, the output is meant to correspond to the rate of firing of a neuron. Outputs are sent down all links emanating from a unit. The inability to conditionally choose where to send what value is one of the major limiting assumptions of the connectionist model, and is motivated by biology. A unit computes its potential and output as a function of its weighted input (via the site values; see below) and internal *state* information. The amount of internal memory is small. The simulations in this thesis use at most two values, a numerical data value to model such phenomena as dynamic thresholds and a discrete state value to model different computational states such as *exhausted*. In the simulations, each unit data structure contains a pointer to the function used to update that unit. Thus, the unit

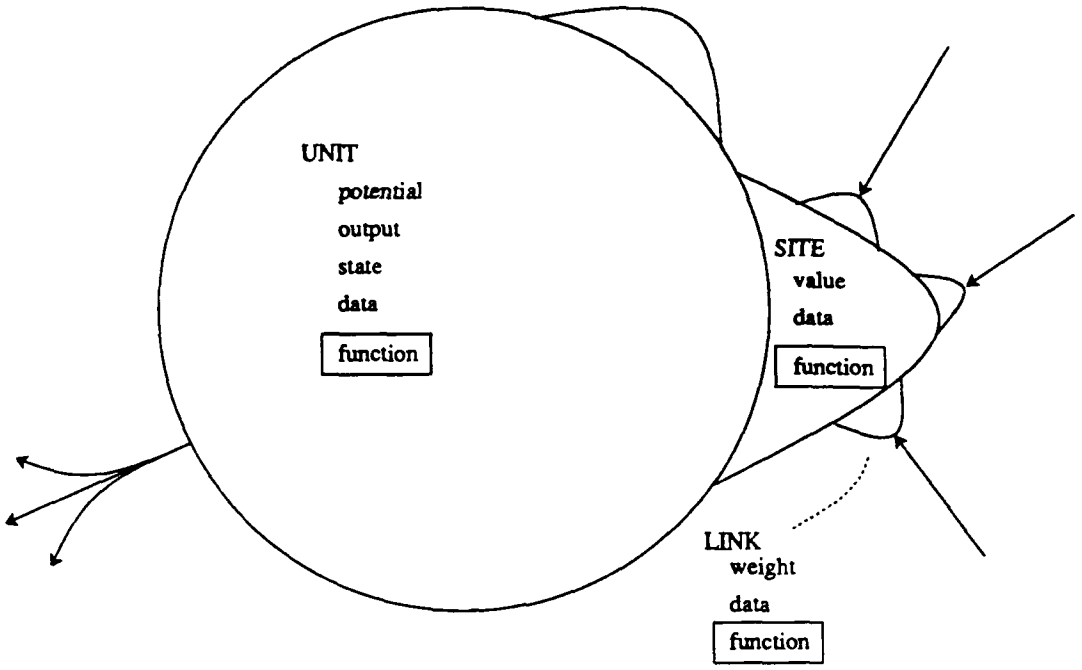

Figure 1: A typical unit.

behavior is individually customizable, though entire populations of units usually share the same function.

To provide greater flexibility, units can have more than one *site* upon which the inputs impinge. Each site combines the inputs, yielding a single number, which the unit uses to calculate the potential and output. For example, a unit might have two sites: one for inputs from area A and one for inputs from area B. The unit function could specify a positive potential if and only if it is receiving strong input at each site. Unlike many connectionist models, the function used to combine the inputs is not fixed; one site could be summing the input, while another is computing the maximum. The function attached to each site has access to all the unit data as well as all the links impinging on that site. It computes the value and data fields of the site. Usually, the value field is used by the unit function and the data field is for "internal" use by the site function. It is possible, for example, to have a dynamic threshold at each site.

Links have *weights*. The weights modify the value transmitted over the link multiplicatively, so a weight of 0.5 would halve the value and reduce the importance of the connection. When the weight is positive, the link is excitatory; when the weight is negative, it is inhibitory. Dynamic modification of link weights is the primary mechanism of adaptation. Associated with each link is a function which assigns a new link weight based upon the current weight, the current value traversing the link and the state of the receiving site and unit. The link data field is used in this thesis to implement decaying link activation.

All the simulations reported in this thesis are synchronous. Each unit computes an output for step t using inputs from other units generated at step $t-1$. In order to speed up the simulations, all numerical values were integers, with the real interval $[0.0, 1.0]$

mapped to the set $\{0, 1, \ldots, 1000\}$.

1.2 Long-Term Goals

Recent progress in connectionist research has been encouraging; networks have successfully modeled human performance for various cognitive tasks [Rumelhart *et al.*, 1986, Waltz and Feldman, 1987]. Still, they are in many respects very limited. Their ability to model large, complex, heterogeneous systems, to follow a chain of reasoning, to simultaneously reason about multiple objects and their complex interrelationships has not been proven. These are the problems which I consider most important and towards which I hope this thesis contributes. I would like to briefly sketch a plan of research which goes beyond this thesis in order to show the relevance of the work done here and give it some context.

The sequence of problems worked on is roughly the following. Since learning is key, the first step is to develop useful learning techniques at the most basic level, using simple, feature-based object descriptions. This is the subject of chapter 3. Much work has already been done in this area. The next step is to extend object descriptions to encompass complex objects with interrelated subparts. Such representations are basic in traditional AI, but are far from simple for connectionist approaches. This is the subject of chapter 4. This is as far as the work in this thesis progressed.

Future work must embed this isolated concept learning in a large, complex knowledge base, using that knowledge to guide learning. There must be hierarchies of concepts and relations [Shastri, 1985], as well as specific facts about the world. Connectionist networks must be capable of controlled, sequential behavior. They must be able to plan ahead and make inferences. These complex processes should be adaptable and should be realized within the connectionist paradigm. These goals may be obvious, but it is important to keep them explicitly in mind. Early work must operate in simple domains. It is all too easy to take advantage of this and rely on methods which could not possibly scale to more realistic problems. The networks described in this thesis, especially the structured object learning network in chapter 4, were designed with this larger context in mind.

1.3 Philosophy of Research

This section explains my philosophy of research. Such matters are often left implicit, but are actually very important in any research program. My major goal is the realization of intelligent processes on computers. This goal is shared by most AI researchers and is what distinguishes them from other cognitive scientists who are more likely to focus on abstract analysis or human performance. My research strategy is progressive refinement of working implementations. A computer implementation has intrinsic value, both scientifically and (ultimately) for society. It requires a model which is completely specified. The process of working out all the details required for an implementation is very useful for gaining a better understanding of the problem, and for designing future, more sophisticated implementations. The first machine to fly could not have been an F16 or

a 747, no matter how many generations of armchair aviators dedicated themselves to the task.

Computer programs can be run and measured objectively. However, it is not possible to experimentally explore the full range of behavior for most tasks requiring intelligence. There are too many possible configurations to test. Any preliminary implementation is full of simplifying assumptions. Determining how techniques will scale is nontrivial. This is why we seek formal analysis. Unfortunately, there is often a conflict between performance and analysis. Analytical techniques work best with models that are simple and mathematically specified. Adding a few hundred lines of special-purpose code may improve performance, but may ruin the analysis. Emphasizing analysis can handicap us. Emphasizing performance can result in ad hoc techniques of no general use.

While the ideal is to achieve both superior performance and analysis, my emphasis is on performance. I hope to avoid ad hoc results, but don't believe this necessitates a precise, mathematical analysis of every step. Nor does such an analysis guarantee generality. The definition of ad hoc is "for a specific purpose, case, or situation."[1] I take the opposite of ad hoc here to be "principled," i.e., there is some general, valid principle guiding the work; the solution looks to the general case, to the long run. This is the standard to which I hold my work.

Of course, a precise analysis and proof of performance is the ultimate achievement in any domain. However, I hold little hope of finding "Maxwell's equations of thought," as Dennett [1988] observes:

> ...in between the mind as crystal and the mind as chaos lies the mind as gadget, an object that one should not expect to be governed by "deep" mathematical laws, but nevertheless a *designed* object, analyzable in functional terms: ends and means, costs and benefits, elegant solutions on the one hand, and on the other, shortcuts, jury rigs, and cheap *ad hoc* fixes.

One of the considerations most affected by the desire for analysis is the computational limitations placed the model: simpler networks are more easily analyzed. There is no universal definition of connectionist networks. Proposed models have a family resemblance, and there have been many different individual specifications, but there is no widely accepted standard. Guidelines can be drawn from many sources, including the brain.

Brain-based constraints are both strong and weak—strong because neurons and their connections inspire the connectionist paradigm; weak because the brain is not yet well understood. There is some confusion in the connectionist literature as to the exactness of the neural modeling. This is confounded by the use of the term "neural network" for everything from matrix multiplication to back propagation. The behavior of actual neurons is quite complex. If connectionist researchers actually want to model neurons, they are doing a poor job. If not, they need to be more exact about the relation between the brain and their model.

Let me be clear about my stand: I make no attempt to model the brain. I am inspired by the ability of the brain to quickly solve hard problems which have resisted algorithmic formulation, but my goal is to create artificial intelligence, not model the

[1] *The American Heritage Dictionary*, Second College Edition.

brain. Units are not neurons, which may be a good thing. By observing nature, one concludes that some billions of neurons are necessary for intelligent behavior. Since there is no near-term hope of efficiently simulating billions of neurons, it may be more productive to keep the analogy loose. On the other hand, as long as connectionist researchers maintain roughly brain-like constraints, it seems likely that they will shed some light on the principles of computing with neurons. This is one reason for using a purer connectionist model than one otherwise might (see below).

At this stage of our understanding, it would be a mistake to overconstrain the model. Each new technique discovered will probably require some new computational mechanism. It is better to add mechanisms when it seems necessary and later evaluate them based on their performance and generality.

In the short run, hybrid approaches may yield the best results [Hendler, 1987]. These are models which borrow from connectionism, but violate its fundamental principles in some way—perhaps by including some non-connectionist modules in a system or by sending complex symbolic messages down links. It would be useful to call a subroutine to create a binding link when appropriate instead of actually implementing a binding network, for example. While exploration of hybrid approaches could be very productive, there are reasons I did not do so. With hybrid models we are no longer addressing the problem of computing with brain-like devices. This is an interesting question in itself: how can neurons compute? The more similar to actual neural networks our models are, the better they answer that question. Also, by forcing ourselves to stick to pure connectionism, we might uncover new and better ways of doing things. Any kind of sequential process presents a bottleneck, especially when we look ahead to the massively parallel hardware on which the networks will run (e.g. [Hillis, 1985]). As long as all the networks completely distribute the computation, avoiding a central interpreter, the potential for massively parallel implementations exists.

1.4 Local Versus Distributed Representations

Finally, I would like to address the apparent split of connectionist researchers into localist and distributed camps. Feldman [1986] addresses this issue. He concludes that neither extremely punctate nor diffuse representations are plausible. In general, however, the most useful representations are compact, with some small number of units representing each concept, and no units representing more than a small number of concepts. I agree with these conclusions; on the whole, my work is more localist than distributed. However, one must be careful not to take too limited a view of what this means.

Many useful properties of distributed representations can be kept in a more compact representation. In choosing a conceptual representation, two principles get us started. First, some basic concepts are represented by a unit. In other words, (some) units have meaning in isolation. Second, more complicated concepts are represented by the collective activation of other concepts (here we pick up some of the advantages of distributed representations), but they also has their own, unique unit (here we pick up the advantages of a local representation). This unit serves as a handle on the rest of the representation. Take the notion of a "grandmother cell." There may be a single

unit dedicated to one's grandmother, but it is connected to other representations of people, women, senior citizens, physical objects, midwesterners, visual images, specific memories, etc. The richness of one's memory of a person relies on many "cells," but it is hard to deal with all this computationally when trying to reason, plan, switch contexts, etc. without some kind of handle. The word "grandmother" acts as a symbolic handle on the concept. An analogous network handle is a reasonable hypothesis.

The psychological phenomenon which intuitively supports handles is chunking. It is not clear how chunking works in a completely distributed representation. As long as something is known to us only as a collection of features, it takes more effort to think about it, but if we can "get a handle" on it, the intellectual burden is lightened. The use of such a handle to avoid cross talk when processing complex objects is discussed in chapter 4.

Note that many of the desirable properties of handle-free distributed representations remain. Property-based, automatic generalization still occurs. When a complex concept is active, the subordinate features are active as well (possibly to varying degrees, depending on the context). When a member of a concept exhibits some behavior, evidential support for this behavior can be attached to the concept as well as to its components. Hinton et al. [1986] point out the similarity of distributed representations and local representations with spreading activation. The use of a handle to bind together a concept is similar to Minsky's [1985] K-Lines.

1.5 The Thesis

The work presented in this thesis does not constitute a single project, but it does all fit the theme of structured learning. The network described in chapter 2 implements a connectionist chart parser. Chart parsing is a well understood technique, but my work is interesting for a couple of reasons. First, it demonstrates a technique for parallelizing parsing at a fine grain. Second, it improves on connectionist parsers in the literature. It is a good example of the power of highly structured connectionist networks. The parser was extended to learn in some situations. Distributing newly learned rules throughout the parsing network proved to be a major difficulty. This, in part, inspired the research on structured learning presented in chapter 4.

Chapter 3 deals with learning simple, structure-less concept descriptions. Many such algorithms exist. The primary motivation is to achieve quicker learning. Two different techniques are presented. The first recruits hidden units representing pairs of inputs using only a global error signal. Layers of pair units represent longer conjunctions of input features. This work was only partially successful. The second technique augments competitive learning [Rumelhart and Zipser, 1985] by providing top-down feedback from concept units. The feedback makes competitive learning sensitive to an input partition imposed externally.

Chapter 4 presents a technique for learning structured objects. The major premise of this work is that the efficient processing of structured objects requires a sequential component to separate the primitives. Control techniques are developed to switch attention between the primitives automatically. The learning algorithm was designed to work in a dynamic environment where the input is not held constant while feedback is

provided. The preliminary implementation presented is simple and has limitations, but it successfully addresses these problems. The final chapter summarizes the work done and presents some directions for future work, including a sketch of the implementation of rule-like inferencing based upon the dynamic representation of structured objects developed in chapter 4.

Chapter 2

PARSING AND LEARNING TO PARSE

2.1 Introduction[1]

My goal in designing a connectionist parser is to be able to systematically build, for any context-free grammar, a network which parses strings in that grammar (within a length restriction). The network must represent the parse tree for the input when finished, and must clearly indicate when there is no parse. Most important, the network must be deterministic and completely general. The goal is not to do cognitive modeling per se, but to provide a technique which might prove useful for natural language understanding and other connectionist applications. Context-free grammars have proven very useful to Computer Scientists. The existence of a fast, simple and relatively efficient connectionist parser may well be of some importance to Cognitive Scientists working with connectionist models.

Several other connectionist parsing schemes can be found in the literature. They all parse context-free grammars, using individual units to stand for the terminal and nonterminal symbols. Given a production such as $S \rightarrow NP\ VP$, there are excitatory connections from NP and VP units to S units which provide bottom-up evidence for the presence of an S, as well as excitatory connections from S to NP and VP providing top-down feedback. When a parse completes, the active units and their connections form a structure isomorphic to the parse tree for the input. I will at times refer to units in a parsing network as parents or children accordingly.

The work of Cottrell [1985] and Waltz and Pollack [1985] encompasses natural language understanding in a much broader sense than syntactic structure alone. Waltz and Pollack do not even attempt to build a general purpose parsing network. Their network is custom built for each input. Cottrell's networks are more general. He even has a program to build the networks from an input grammar. Unfortunately, the network does not always find the correct parse.[2]

[1] This bulk of this chapter first appeared as [Fanty, 1985]. An abbreviated version appeared as [Fanty, 1986b].

[2] This can be an advantage from the viewpoint of cognitive modeling if the errors made resemble those made by humans.

In both models, contradictory interpretations of the input are mutually inhibitory. When the network stabilizes, a single consistent interpretation should be isolated. Which one wins depends on how much semantic and syntactic support each receives from the rest of the network. A parse proceeds by activating the input, e.g. the unit for "the" in position one, the unit for "man" in position two, etc., and letting the network run until it settles into a consistent configuration representing the parse of the input with all ambiguities resolved.

The work of Selman and Hirst [1985] is nearer my own in ambition: their goal is a general purpose parsing scheme which will perform correctly for any input. They do not consider nonsyntactic influences. Mutually inhibitory binder units connect nonterminal units to all the subtrees they might dominate, each binder representing a different production of the nonterminal. Likewise, binder units connect a unit to all units which might dominate it. They use a variation of the Boltzmann machine [Hinton *et al.*, 1984] computational scheme. The units representing the input are clamped on. The others execute asynchronously, turning on or off probabilistically based on how much excitation and inhibition they are receiving. Simulated annealing [Kirkpatrick *et al.*, 1983] is used to settle the network into a state where the active units are mutually reinforcing to a large degree. The best possible state is one which represents a legal parse. In order to reach this optimal state with high probability, the network must settle gradually. Selman and Hirst used 24,000 updates for each unit.

My network is deterministic, fast, guaranteed to work for all inputs of any context-free grammar, and conceptually very simple. Contradictory parses do not inhibit each other (but see section 2.3); all possible parses proceed in parallel. This requires a large number of units—typically tens or hundreds of thousands (see below).

The remainder of this chapter is organized into four parts. Section 2.2 describes the network in detail, giving an algorithm for generating it from a given CFG. An exact analysis of the network's complexity is presented, along with some ways of trimming its size. Section 2.3 touches on some methods of disambiguating, i.e. choosing a unique parse tree for ambiguous input. Section 2.4 gives a cursory account of parsing near-miss input, i.e. input which is almost grammatical. Section 2.5 gives a detailed description of how productions can be learned dynamically in some circumstances. These latter sections are intended to explore the flexibility of the parser as well as test some connectionist learning techniques. They do not provide an account of language acquisition.

All networks have been implemented. Simulation traces are included below.

2.2 The Network

2.2.1 Structure and function

The strategy used closely parallels that of the CYK parser [Hopcroft and Ullman, 1979]. The network contains units representing the terminals and nonterminals of the grammar (several for each, in fact) as well as match units which will be explained below. The units are best thought of as organized into a table, with the columns representing starting positions in the input string, and the rows representing lengths. There is a unit for each terminal symbol in each position of row one. There is a unit for each nonterminal at every position in the table (potentially; see section 2.2.2). Terminal units are activated

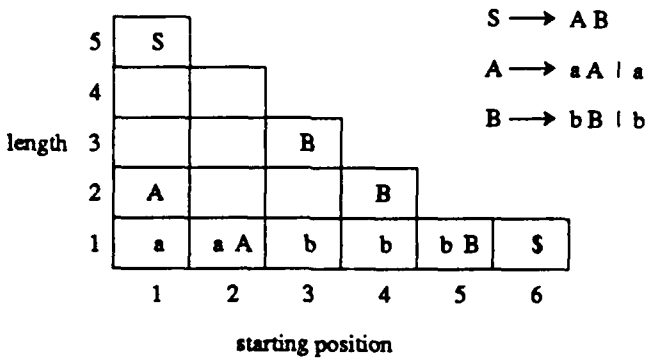

Figure 2: Parsing the input aabbb. Only the relevant units are shown.

by some outside source and represent the input to the parser. A nonterminal unit will become active if other units representing the right-hand side of one of its productions, and having appropriate starting positions and lengths, are active. The parse proceeds in a bottom-up fashion. Figure 2 illustrates the parsing of the string *aabbb* for the grammar shown. The terminal symbols are activated as input. The A unit in the first row becomes active because of input from the a unit in the same position and the production $A \rightarrow a$. The A unit in (2,1)—row two, column one—becomes active because of input from both the a unit in (1,1) and the A unit in (1,2) and the production $A \rightarrow a\ A$. The B units activate in a similar fashion. The S unit becomes active because of input from A in (2,1) and B in (3,3). Because there is an active start symbol in column one whose length is the same as the length of the input, the input is accepted. In order to mark the end of the input, a special \$ unit becomes active at the end of the input string.

The active units represent the parse tree. Of course, there will generally be many units which become active but don't represent a final parse tree. In the above parse, there will be an active A unit in (1,1), for example. This extra activation will not affect the ability of the network to correctly recognize legal inputs, but a second, top-down, pass of activity is necessary in order to pick out only those units which participate in a complete parse—if the string is ambiguous, more than one parse will stay active (see section 2.3 for possible modifications). The top-down pass works as follows: when an active unit representing the start symbol beginning in position one and of length n receives input from a \$ symbol in position $n + 1$, it becomes hyperactive. This unit is the root node of all parse trees. It passes this hyperactivity down to all units which form part of one of its completely recognized productions. They pass the activity down in turn until it reaches the bottom. Only the hyperactive units represent a parse. In order to better distinguish the two levels of activity, a unit activated during the first bottom-up pass will be called "primed," and a unit active after the the final pass will be called "on." In our simulations, primed units have a potential and output of 5; on units have a potential and output of 10.

A more detailed account of the network follows. There are three kinds of units: nonterminal units, terminal units and match units. Each unit has two sites where an incoming connection might be made. One is for bottom-up input. Enough input to this site will prime the unit. The other site is for top-down input. Every pair of units

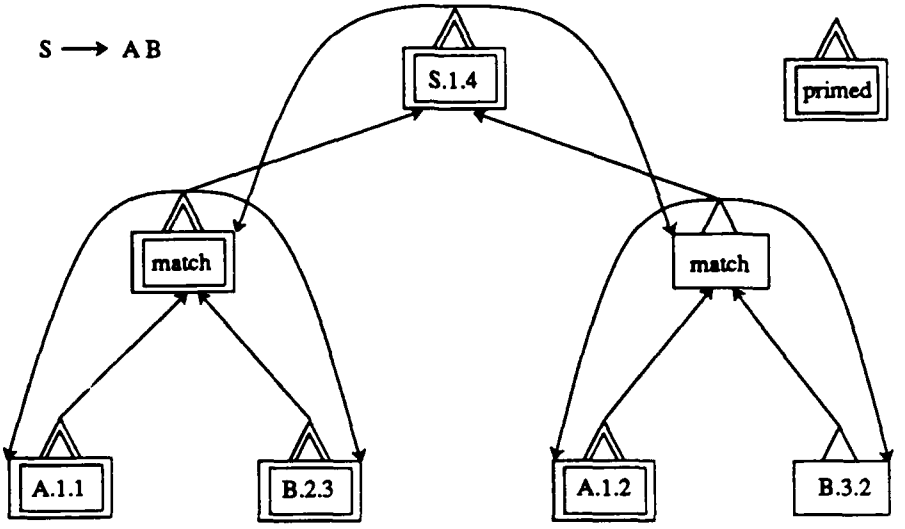

Figure 3: Two match units and their connections

with a bottom-up link between them also has a top-down link. If a primed match or nonterminal unit receives input to the top-down site from an on unit, it will turn on. The site functions are described below, and are given exactly in appendix two.

The terminal units are the simplest. They are primed by some outside source and represent the input to the parser. They are all on row one because they must be of length one. One per starting position should be primed in order to represent an input, although more than one might be activated in the case of input ambiguity (see section 2.3). The $ units are a special kind of terminal unit. One should be turned on in the position following the input.

The match units are used to represent the various instances of productions for nonterminal units. They receive bottom-up inputs from units representing the symbols on the r.h.s. of their production. The starting positions and lengths of these units must be consistent with the nonterminal unit being served and with each other. Each nonterminal unit has a separate match unit for each allowable combination of lengths of each production (see figure 3). The bottom-up links to the match units are weighted so that all the connected units must be primed before the match unit becomes primed. The bottom-up inputs to a match unit are processed with a filtered-sum function: The sum of the inputs is taken, but each input is allowed to contribute at most some fixed amount. This seems the most natural and flexible way of fixing the following problem.

Suppose match unit X represents a production with three nonterminals on the r.h.s. Two of the three nonterminal units are primed. With weights of 2/3 on the links, the input to the unit is

$$5 \times \frac{2}{3} + 5 \times \frac{2}{3} + 0 \times \frac{2}{3} = 6.67$$

which is below the threshold of ten, as desired. Now suppose the first nonterminal unit turns on because of top-down activation from somewhere else. The input to the match unit is now

$$10 \times \frac{2}{3} + 5 \times \frac{2}{3} + 0 \times \frac{2}{3} = 10.0$$

which causes it to become primed falsely. This is prevented by limiting each bottom-up input's influence to 5 × weight.

A nonterminal unit receives bottom-up input from its match units. If any of them become primed, this means one of the productions has been realized, so the nonterminal unit becomes primed. A nonterminal unit responds very simply to bottom-up input from its match units: if any are primed, it becomes primed and provides bottom-up input to match units above it.

If the priming eventually reaches a start-symbol nonterminal unit in column one, row n and the input is of length n, then there exists a legal parse of the input. The root unit will be receiving "top-down" on input from the $ unit in row $n + 1$ used to mark the end of the input. This turns on the root unit, which now provides top-down on input to its match units. Any which were primed turn on, and provide top-down on feedback to all connected units. In this way the parse tree(s) will be turned on from the top down. In order to achieve correct behavior, the units must respond to top-down input in the following way. If the unit is already primed and the top-down input is at the on level, then the unit turns on. If the unit were not required to first be primed, then an on nonterminal unit would turn on all its match units, even those representing quiet production instances. Match units have only one input to their top-down site, so they can simply respond to it in a thresholded manner. Nonterminal units will typically have many inputs to their top-down site. Simply summing these inputs would result in incorrect behavior, as input from two primed match units will have the same sum as input from a single on match unit. To differentiate the two cases, nonterminal units take the maximum input as the value of the top-down site. This must be on to have any effect.

When primed terminal units receive on input at their top-down site, they turn on as well. It would be possible for the network to detect when the parse is complete by sensing the presence of an on terminal symbol in every column up to the length of the input, though this was not implemented. If we exclude productions with a r.h.s. of length one, the parse will complete in at most 4 × input − length steps. The network could reject input by timing out.

The network can be turned off by taking away the external input to the terminal units (including $). This will remove bottom-up and top-down activation from the network: the source of all bottom-up activation is the terminal units; the source of all top-down activation is the $ unit acting through the root node.

2.2.2 Network construction

The parsing network for a grammar is built by a program. The algorithm appears in Fanty [1985]. The strategy is to work bottom up, first placing each terminal in each position of row one. Since the productions of a unit on row n depend only on units in rows $\leq n$, the network is built in a single bottom-up pass through the table. For each combination of lengths of each production, the appropriate units are looked up in the table. If they exist, a match unit for that production is created and the appropriate links are made. If none of the productions of a nonterminal in a given location can be realized, the nonterminal unit is not made. If, for example, the nonterminal B could only generate strings of length three or more, then no nonterminal units of the form B.n.1

or B.n.2 would be created, and there would be no match units in charge of productions looking for such units.

For simplicity, ϵ-productions are not allowed, which is not too limiting, as a grammar with ϵ-productions can always be translated to one without. They could easily be added if desired. If there are productions of the form $X \rightarrow Y$, where Y is a single nonterminal, then Y units must be processed before X units on each row.

2.2.3 Complexity of the network

In order to facilitate the discussion, I introduce the following:

L = maximum length of input string
T = set of terminals in the grammar
N = set of nonterminals in the grammar
$\pi(n)$ = set of productions of nonterminal n
$\nu(p)$ = list of nonterminals in production p (may be repeats)
$\tau(p)$ = list of terminals in production p (may be repeats)

The number of nonterminal and terminal units is reasonable. In the worst case there are

$$\frac{L(L+1)}{2}|N| + L|T|$$

of them. The number of match units is significantly larger, however. In the worst case there are

$$\sum_{r=1}^{L} \sum_{c=1}^{(L+1)-r} \sum_{n \in N} \sum_{p \in \pi(n)} \begin{pmatrix} (r-1) - |\tau(p)| \\ |\nu(p)| - 1 \end{pmatrix}$$

match units. The sum represents, for each row and column, for every production of every nonterminal, all the possible combinations of constituent lengths in an expansion of that production. The quantity $(r-1) - |\tau(p)|$ choose $|\nu(p)| - 1$ represents the number of different assignments of lengths to constituents which will sum to the desired length. The reasoning is as follows. How many ways can three constituents sum to ten? There are two boundaries between constituents to be chosen, and nine different locations to choose from: $(10-1)$ *choose* $(3-1)$. The quantity $|\tau(p)|$ appears in the first term because terminals reduce by one the space into which the nonterminals may expand.

For example, if $L = 15$, $N = 10$, $|\pi(n)| = 5$ and $|\nu(p)| = 3$ for all nonterminals and ignoring terminals, then the network will have at most 1,200 nonterminal units and 153,750 match units. This figure can best be improved be keeping $\nu(p)$ small (by putting the grammar in Chomsky normal form, for example). If $|\nu(p)| = 2$ in the above example, the number of match units will be at most 34,000. The size of the network is $O(n^{m+1})$, where n is the length of the network and m is the number of nonterminals on the right-hand side of the productions. Thus, it may be desirable in practice to limit the number of nonterminals on the right-hand side of a production to two. If this is done, the size of the network will be $O(n^3)$.

For the following grammar, taken from Selman and Hirst [1985], for inputs of length up to 15, the number of nonterminal units is 570 and the number of match units is 2,040.

S → NP VP	NP → determiner NP2
S → VP	NP → NP2
VP → verb	NP → NP PP
VP → verb NP	NP2 → noun
VP → VP PP	NP2 → adjective NP2
PP → preposition NP	

The number of units may be significantly smaller than the worst case if several nonterminal units cannot generate strings of all lengths (especially short lengths). The network described above would have 30 more nonterminal units and 210 more match units if PPs could generate strings of length one. A better example of the potential savings is provided by the following grammar which has the same number of nonterminals and productions as the preceding grammar, but no nonterminal can generate strings of length one.

S → D B	D → c A
S → B D	D → A
B → a D	D → D C
B → B C	A → e d
C → b D	A → e A

For inputs of length up to 15, the number of nonterminal units is 458 and the number of match units is 1561. If nonterminal units for each nonterminal were placed at each location of the table, the number of nonterminal units would be 600 and the number of match units would be 2794.

The simulations run in $O(n)$ time, where n is the length of the input. Multiplying the execution time by the number of units (with the length of the r.h.s. of productions limited to two), gives a total of $O(n^4)$ computation steps. The serial execution time for parsing is $O(n^3)$ for straightforward parsing algorithms and about $O(n^{2.5})$ for the asymptotically best algorithm so far. This means that the network is bigger/slower than the best we could expect by about a factor of n. Intuitively this is because the algorithm is not completely parallelized. The activity must work its way up from the bottom serially.

2.2.4 Implementation using binary linear threshold units

It is possible to build an equivalent network using only simple, single-site, linear threshold units which have only two levels of activity: on and off (1 and 0). The transformation is simple. Replace every unit in the original network with a pair of units: one for the bottom-up pass, and one for to the top-down pass (see figure 4). The bottom-up unit being active corresponds to the original unit being primed; the top-down unit being active corresponds to the original unit being on. The bottom-up inputs to match unit pairs will be weighted as before to require them all to be on. They are no longer filtered as they only come from other bottom-up units. The top-down unit takes a sum rather than a maximum of its inputs as it only receives input from other top-down nodes. In order to require that the pair be primed before turning on, the weight on the link from the bottom-up unit to the top-down unit equals the sum of the weights on the other

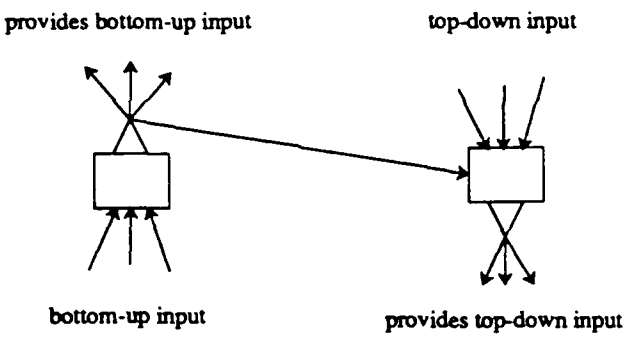

Figure 4: Two simpler units can do the job of one more complex unit.

inputs to the top-down unit, and the threshold on the top-down unit is set so that it must receive input from the bottom-up unit and at least one top-down input. When the parse completes, the active top-down units represent the parse tree.

2.2.5 Informal proof of correctness

In order to establish the correctness of a network as described above, we need only show three things. (Assume the grammar has no ϵ-productions. All units begin off except those terminal units representing the input, which are primed.)

1. A nonterminal unit A.m.n will become primed if and only if other terminal and nonterminal units spanning m through $m + n$ in exactly the order of one of A's productions are primed.

   We will say that these other terminal and nonterminal units satisfy one of A's productions. A will become primed if and only if one of its match units becomes primed. These are the only units with connections to its bottom site, and only input to this site will cause an off unit to become primed. Each match unit corresponds to one of A's productions. It has an input from a unit corresponding to each symbol in the production in such a way that positions m through $m + n$ are spanned exactly. The weights on the inputs to the match units are such that they must all be primed in order for the input to exceed the threshold. Each input is filtered so that no single input can contribute more than its share. Every possible satisfaction of each of A's productions has a corresponding match unit.

2. When the input is of length n, the first nonterminal unit to turn on (if any do) will be S.1.n, and it will turn on only if it is first primed. (S is the start symbol.)

   In order for a nonterminal unit to turn on it must first be primed and then receive on input to the top-down site. Since input from an on unit is required to turn another unit on, the first unit to turn on must receive top-down input from \$.(n+1).1, the only unit on from the beginning. The only unit with such a connection is S.1.n.

3. A nonterminal unit—other than the first to turn on, as in (2)—will turn on if and only if it is first primed and is one of the units satisfying a nonterminal unit which turned on previously.

Following (2), we need only show that a primed nonterminal unit will receive input to its top-down site from an on unit if and only if it helps satisfy some production of an on nonterminal unit. Except for input from the on **$** unit, which has just one connection to the root node, nonterminal units receive input to their top-down site only from match units to which they contribute. A match unit will turn on only when it is first primed (production satisfied) and its parent nonterminal unit turns on. All inputs to a primed match unit contribute to the satisfaction of its production instance.

2.2.6 Simulation results

A network for the grammar in section 2.2.3 was built and simulated with the following input: *det noun verb det adj adj noun*, which corresponds to sentences such as *The man kissed the tall attractive woman.* The results of the simulation are given in table 1. Only units which have non-zero potential are shown. Match units are omitted in order to make the table more readable. After 26 steps, the network is stable. Those units with a potential of 10 represent the (unique) correct parse.

Table 2 shows the results of simulating *noun verb det noun prep noun prep det noun*, e.g. *John hit the man with Tom with a hammer.* This sentence is ambiguous in many ways. Notice the overlapping constituents, such as the *PP* from 5 to 9 (PP.5.5) and the *NP* from 3 to 6 (NP.3.4). The match units provide enough information to distinguish the various parses, but if the match units are invisible externally, the state of the network does not make sense. In any case, it may be desirable to select only one parse. This is the topic of the next section.

2.3 Disambiguation

A lot of local disambiguation happens naturally because some interpretations do not participate in a complete parse tree and, thus, never turn on. This could account for word sense disambiguation in many cases. For example, to parse the sentence *The man walked on the deck*, both noun and verb would be primed in position six, but only noun would turn on, as there is no complete parse using the other interpretation.

When the input is syntactically ambiguous, however, more than one parse tree will be on simultaneously. The parse can be made unambiguous by allowing only one match unit (i.e. production) per nonterminal unit to remain active. The simplest way to do this is to order the match units of each nonterminal and add inhibiting links from each to those of lesser rank. Any inhibiting input from a superior match unit would prevent activation. Only the highest ranking primed unit would stay primed. The match units would need to be ranked not only according to which production of the grammar they represent, but also according to the combination of lengths of their production. The ranking would not need to be consistent throughout the network. One production could dominate another only for short lengths or towards the beginning, for example.

The above scheme was not implemented; however, a different scheme was. In this scheme, each match unit inhibits all the other match units belonging to the same nonterminal unit. An off match unit receiving inhibiting input will not become primed. Thus this scheme prefers shallower parse trees.

Table 1: Simulation of *det noun verb det adj adj noun.*

Unit name	Potential after 1 step	Potential after 13 steps	Potential after 26 steps
det.1	5	5	10
noun.2	5	5	10
verb.3	5	5	10
det.4	5	5	10
adj.5	5	5	10
adj.6	5	5	10
noun.7	5	5	10
$.8	5	5	5
NP2.2.1	0	5	10
NP.2.1	0	5	5
VP.3.1	0	5	5
S.3.1	0	5	5
NP2.7.1	0	5	10
NP.7.1	0	5	5
NP.1.2	0	5	10
S.2.2	0	5	5
NP2.6.2	0	5	10
NP.6.2	0	5	5
S.1.3	0	5	5
NP2.5.3	0	5	10
NP.5.3	0	5	5
NP.4.4	0	5	10
VP.3.5	0	5	10
S.3.5	0	5	5
S.2.6	0	5	5
S.1.7	0	5	10

Table 2: Simulation of *noun verb det noun prep noun prep det noun.*

Unit name	Potential after 13 steps	Potential after 26 steps	Potential after 26 steps (disambiguated section 2.3)
NP2.1.1	5	10	10
NP.1.1	5	10	10
VP.2.1	5	5	5
S.2.1	5	5	5
NP2.4.1	5	10	10
NP.4.1	5	5	5
NP2.6.1	5	10	10
NP.6.1	5	10	10
NP2.9.1	5	10	10
NP.9.2	5	5	5
S.1.2	5	5	5
NP.3.2	5	10	10
PP.5.2	5	10	10
NP.8.2	5	10	10
VP.2.3	5	10	10
S.2.3	5	5	5
NP.4.3	5	5	5
PP.7.3	5	10	10
S.1.4	5	5	5
NP.3.4	5	10	5
NP.6.4	5	10	5
VP.2.5	5	10	10
S.2.5	5	5	5
PP.5.5	5	10	5
S.1.6	5	5	5
NP.4.6	5	5	5
NP.3.7	5	10	5
VP.2.8	5	10	10
S.2.8	5	5	5
S.1.9	5	10	10

Because the simulations are synchronous, multiple match units will prime simultaneously whenever their subtrees have equal depth. When this happens, the inhibition must be gradual, or the match units will turn each other off. This would allow them all to come back on the following step and cycle in this manner indefinitely. The following behavior reliably yields a single winner. The inhibiting weights between the match units vary randomly between -0.5 and -1.0 exclusive. A primed unit receiving inhibition will lower its potential by an amount equal to the strongest inhibiting input. When the potential gets to zero, it turns off. When only one match unit is left, the lack of inhibition allows it to gain its full primed potential of five. For example, suppose two match units become primed at the same time and that match unit M1 inhibits M2 with weight -0.6 and M2 inhibits M1 with weight -0.65. The following is a trace of their behavior in one step increments (with minor arithmetic errors):

M1	5.0	1.75	0.45	0.0	0.0
M2	5.0	1.99	0.94	0.67	5.0

M2 has a higher potential after the first round of inhibition because it is more weakly inhibited. Now M2 is receiving inhibition equal to -0.6×1.75 and M1 is receiving inhibition equal to -0.65×1.99. M2's domination of M1 is increasing. There is no way for M1 to push M2 under 0 since its potential is less than M2's and its inhibition is less than its potential. With more than two match units on, the interactions become more complicated. If the inhibiting inputs were summed, it would be possible for every match unit to go off after one step and oscillate as described above. This is why the total inhibition is equal to the strongest single input.

If the inhibitory weights between match units were adjusted dynamically according to how often the match unit came on, then the network could learn to prefer more common interpretations. It might also be possible to use nonsyntactic information to affect the preferred parse with external contributions to the inhibition. This was not implemented.

The last column of table 2 shows a simulation of the ambiguous sentence from the previous section using the disambiguation scheme just described. A single unambiguous parse tree results. No match units battled, as the ambiguities involved parse trees of different depths; the first on won by default. For example, in deciding between the two productions $VP \rightarrow verb\ NP$ and $VP \rightarrow VP\ PP$, the latter always wins because its parse tree is shallower. I make no claims about the adequacy of this scheme. I present it as a demonstration of how disambiguation could be done in this model. Other strategies are possible as well.

2.4 Parsing Near-Miss Input

It is sometimes desirable to provide a reasonable parse of input strings which are not in the language defined by the grammar—ungrammatical but still understandable natural language utterances, for example. The parsing network described above is too rigid to do this effectively if some of the input is missing or there is extra input. But if the source of the ungrammaticality is simply the substitution of incorrect input of the same length as some correct input, then it can be made to complete an approximate parse by

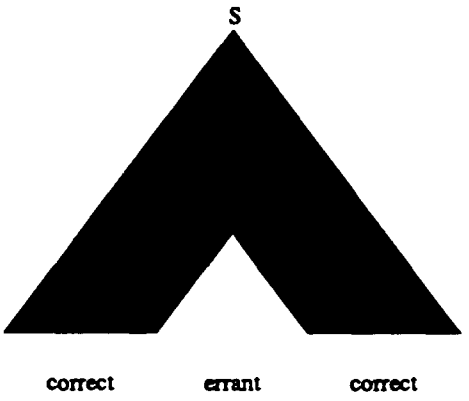

Figure 5: A near-miss parse

having match units become partially primed proportional to the closeness of the match between the input and the production.

The network implemented extends the disambiguating network described previously. Match units whose productions are partially satisfied may have a potential between 0 and 2. This range was chosen so that inhibition from a partially-primed match unit will always be less than inhibition from a primed match unit. The weights on inhibiting connections range from -0.5 to 1.0. A primed match unit will always deliver inhibition of -2.5 or less; a partially-primed match unit will always deliver inhibition of -2.0 or more. Primed match units always inhibit partially-primed match units. Partially-primed match units compete with each other much as primed match units do (see section 2.3), except the inhibition from a partially-primed match unit is not strong enough to prevent the priming of other match units. This is necessary, as a match unit may become partially-primed before one of its brothers becomes fully primed. The strongest partially-primed match unit may be beaten out by weaker partially-primed match units because of the random variation in the strength of the inhibiting weights.

Because inhibition from a partially-primed match unit does not prevent other match units from becoming partially-primed, care must be taken to prevent the re-priming of a partially-primed match unit which has just been defeated. Unless it subsequently receives additional input and becomes fully primed, a defeated partially-primed match unit will stay off.

The permissiveness of the network can be adjusted by setting a minimum threshold for partial priming. In the simulations run, the cutoff was 3 (an input of 10 represented complete satisfaction of the production). The potential of a partially-primed match unit is equal to its total input divided by 5. Some of the key steps in a simulation of the ungrammatical input det det noun verb noun are given in table 3. The grammar used was the one in section 2.2.3. Only the relevant units are included. As before, many units not in a complete parse will become primed. Even more become partially-primed. All the non-match units which eventually turn on are shown. The partial-priming of NP.1.3 after step 3 is due to input from det.1; after step 7, it (actually, its match unit) is also receiving some input from the partially-primed NP2.2.2 so its potential increases to 1.27. Eventually, the partial priming reaches the root node S.1.5. What happens next differs from previous networks. In this network, the end-marker, $.6, must have

Table 3: Near-miss parse of *det det noun verb noun.*

	Potential after step							
Unit	0	3	7	10	14	15	17	23+
det.1	5.0	5.0	5.0	5.0	5.0	5.0	5.0	10.0
det.2	5.0	5.0	5.0	5.0	5.0	5.0	5.0	5.0
noun.3	5.0	5.0	5.0	5.0	5.0	5.0	5.0	10.0
verb.4	5.0	5.0	5.0	5.0	5.0	5.0	5.0	10.0
noun.5	5.0	5.0	5.0	5.0	5.0	5.0	5.0	10.0
$.6	5.0	5.0	5.0	5.0	10.0	10.0	10.0	10.0
NP2.3.1	0.0	5.0	5.0	5.0	5.0	5.0	5.0	10.0
NP2.5.1	0.0	5.0	5.0	5.0	5.0	5.0	5.0	10.0
NP.5.1	0.0	0.0	5.0	5.0	5.0	5.0	5.0	10.0
NP2.2.2	0.0	0.0	1.05	1.05	1.05	1.05	1.05	10.0
VP.4.2	0.0	1.05	5.0	5.0	5.0	5.0	10.0	10.0
NP.1.3	0.0	1.05	1.27	1.27	1.27	1.27	10.0	10.0
S.1.5	0.0	0.0	0.0	1.32	1.32	10.0	10.0	10.0

potential equal to 10 in order to provide sufficient top-down feedback to turn on the root node. Five steps are required to reach that level. This is to give complete parses a chance to reach the top. Once a partially-primed match unit turns on, it is too late for a fully primed match unit to inhibit it. When the parse completes, the section of the input which does not fit into the parse remains primed.

In a simulation with input *det det det verb det det det*, no parse completed. NP1.3 and VP4.4 were both partially primed, but their combined input was less than the threshold for S.1.7's match unit.

2.5 Learning New Productions Dynamically

The near-miss network described above has been modified to learn new productions dynamically. The circumstances under which it is capable of learning are depicted in figure 6. After a near-miss parse, there will be a gap in the parse tree where some constituent was "expected" but not found. If the gap can be parsed as one or two constituents, then a match unit representing the new production will be recruited [Feldman, 1982] ($A \rightarrow B\ C$ in figure 6).

This mechanism cannot explain the acquisition of a grammar from raw data. For one thing, no new nonterminals are learned, which is especially limiting given the restriction on production length (see below). The mechanism can sometimes account for new rules composed of known constituents. The real purpose of this section is to explore the flexibility of the parsing network. I do not claim to have an adequate mechanism for grammar acquisition.

2.5.1 Local learning

This section describes the recruitment of a match unit to represent a production instance. The production will not be recognized elsewhere in the network. In order to make the problem more tractable, the right-hand side of productions must be of length

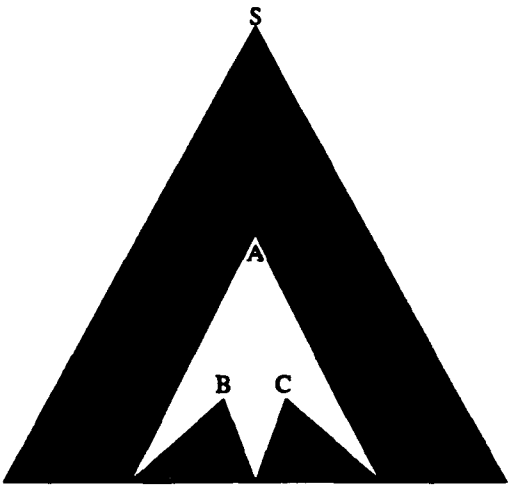

Figure 6: The production $A \rightarrow B\ C$ will complete the parse.

one or two. I will call such productions type one and two respectively. For any context-free grammar, there is a weakly equivalent grammar satisfying this restriction. Each bottom-up input to a match unit now goes to its own site. Accompanying each nonterminal unit is a single learn unit and some free match units which do not yet represent a production. Fixed match units represent production instances as before except for the presence of two bottom-up sites for type two productions. An unprimed nonterminal unit is turned on from above only when an instance of the nonterminal is expected but not found. This turns the learn unit on, which enables the free match units. If the input in question can be parsed as one or two constituents, then some match unit will be recruited to represent this production instance.

The additional bottom-up sites are to enable free match units to detect when they are receiving bottom-up input from a potential production. Each free match unit can learn only production instances with some fixed combination of constituent lengths or division. For example, the free match unit in figure 7 can learn productions with two constituents, the first of length four and the second of length six. Notice that any combination of one input to the bottom-right site and one input to the bottom-left site constitutes a legal production.

Figure 7 depicts the setup of a free match unit just before learning occurs. The free match unit is receiving bottom-up input to each site from exactly one unit. It has not yet primed because it is in a free state; it requires additional input from the learn unit before responding. The nonterminal unit B.3.10 is on but no match unit is primed. This will cause the learn unit to come on. The learn unit requires on input from B.3.10 and is inhibited by the match units. Once the learn unit comes on, the free match unit will become highly active briefly. I will call this state excited. It now inhibits the learn unit, whose job is done. If more than one input per bottom-up site had been active, the free match unit would not have responded.

Three changes now occur which transform the free match unit into a fixed match unit. First, the unit no longer enters the free state in which input from the learn unit is required. Second, the weights on all inactive bottom-up links are zeroed. The

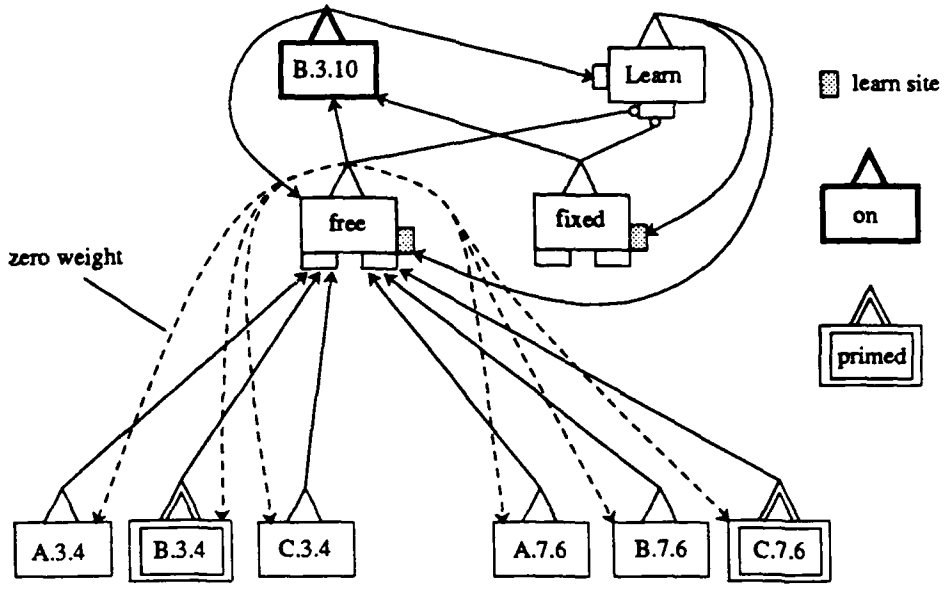

Figure 7: Configuration of free match unit just before learning.

match unit now responds only to the pair learned. Third, the top-down links to the learned constituents are given positive weights. It is the need for this weight change which necessitates the special excited activity of a match unit which has just learned a production. Weight change occurs at the destination unit. The only way for B.3.4 and C.7.6 (see figure 7) to know to increase the weight on the top-down links from the match unit is from this level of activity. The other, non-primed units receiving top-down excited input do not change the weights. Because of the excited input, B.3.10, B.3.4 and C.7.6 become momentarily excited as well. Soon, all excited units settle to an on state and the parse completes. The situation after learning is shown in figure 8.

It is possible for free match units in separate divisions to each learn a production at the same time. This ability is necessary for global learning in the next section. Since all match units inhibit each other, it is necessary to suppress this inhibition during learning. Because of this, all interpretations of ambiguous input will be learned if they are in different divisions and none will be learned if they are in the same division. If there is more than one match unit representing a division, only one should learn a new production. This can be achieved by ordering the free match units of a division and putting strong inhibitory connections from the earlier match units to the learn site of the later ones.

Table 4 shows a simulation of a local-learning network for the simple grammar

$$S \rightarrow A\ B \quad A \rightarrow a\ a \quad B \rightarrow b$$

The production instance being learned is B.3.1 → b.1. After step 13, the near-miss parse has activated B.3.1 even though it was never primed. The learn unit and the free match unit take a couple of steps to come on. B.3.1 and, after step 15, the learn unit are decaying. If learning does not happen, they will turn off eventually. Notice that a.3

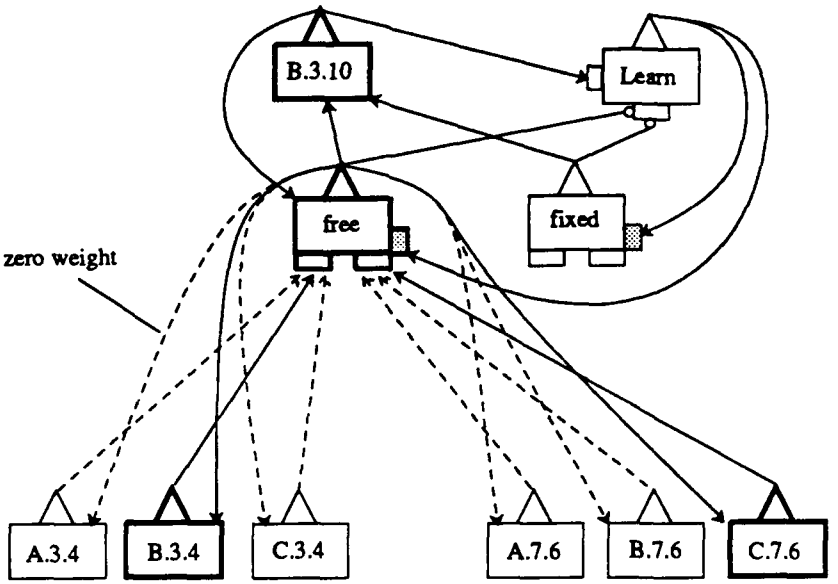

Figure 8: Configuration of match unit just after learning.

Table 4: Demonstration of local learning.

	Potential after step								
Unit	12	13	14	15	16	17	18	29	20
a.1	5	5	5	10	10	10	10	10	10
a.2	5	5	5	10	10	10	10	10	10
a.3	5	5	5	5	5	5	15	15	10
$.4	10	10	10	10	10	10	10	10	10
B.3.1	0	10	9	8	8	15	15	15	10
learnB.3.1	0	0	8	15	14	0	0	0	0
match	0	0	0	3	15	15	15	10	10
A.1.2	5	10	10	10	10	10	10	10	10
S.1.3	10	10	10	10	10	10	10	10	10

Table 5: Same input as table 4, but after learning.

	Potential after step				
Unit	6	7	9	10	11
a.1	5	5	5	5	10
a.2	5	5	5	5	10
a.3	5	5	5	5	10
$.4	10	10	10	10	10
B.3.1	5	5	10	10	10
learnB.3.1	0	0	0	0	0
match	5	5	5	10	10
A.1.2	5	5	10	10	10
S.1.3	5	10	10	10	10

becomes excited two steps after the match unit does. The delay is caused by the need to increase the weight on the link from the match unit to a.3. Table 5 highlights a parse of the same input after learning.

2.5.2 Global learning

This section describes an extension to the above scheme which distributes a production instance learned locally throughout the whole network. Although the mechanism is different, the idea of using a central template to program other representations was inspired by McClelland's [1985] Connection Information Distributer. I will first give a high-level description of what happens, then provide a more detailed description of the implemented network. There is a single, global representation of each production. When learning occurs locally, the production learned is noted in the global store. After the network calms down, it enters a special learn state, which limits the spreading activation. To achieve global learning, the units involved in the production are activated throughout the network by the global representation in such a way that the local learning mechanism of the previous section burns in all the production instances. Unlike McClelland's scheme, no weights are shipped. The transfer of learning is a one-time event that results in permanent changes remotely.

When every instance of some production in the network is active and learning simultaneously, a great deal of care must be taken to insure cross-talk does not occur. A single unit, such as NP.3.4, may be both the parent and son for the same production. In order to distinguish the various roles a unit may play, a separate unit is used for each. To facilitate this, the grammar must be in Chomsky normal form, which means that all productions have the form

$X \rightarrow Y\ Z$ or $X \rightarrow$ a, where X, Y and Z are nonterminals and a is a terminal. Any context-free grammar can be converted to a weakly equivalent grammar in Chomsky normal form [Hopcroft and Ullman, 1979]. For such a grammar, there are only three roles a nonterminal can play: parent, left-son and right-son. Accordingly, the job previously done by nonterminal units is now done by a trio of units, one for each role of the nonterminal. Figure 9 shows the setup for the production instance A.3.4 → B.3.1 C.4.3.

Normally, the three nonterminal units pass on activation. Bottom-up activation is

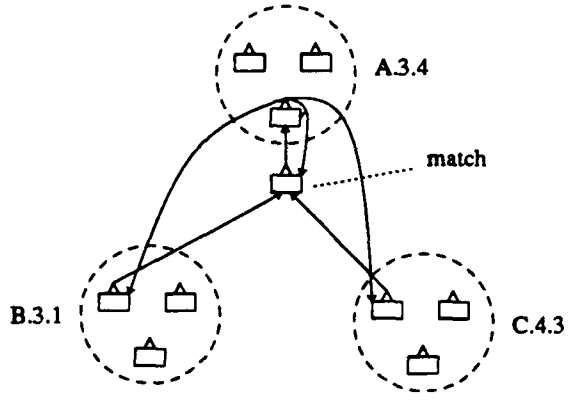

Figure 9: Production instance A.3.4 → B.3.1 C.4.3.

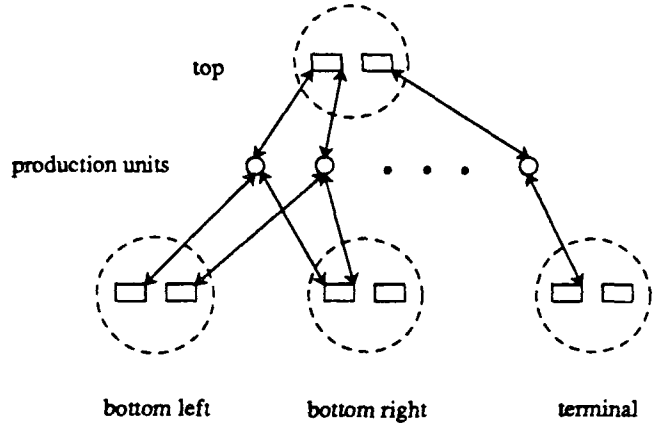

Figure 10: Global production templates.

spread from the match unit to the parent unit to the two son units which are connected to other match units. Top-down activation comes from one of the son units and goes through the parent unit to the match unit. However, the spread of activation between parent and son units for a nonterminal group is blocked when the network is in a global learn state. In the networks implemented, this is done by having the activation go through pass units which are inhibited by a GlobalLearnUnit (see figure 11). When the network is not in a global learn state, it behaves just like the one described in section 2.5.1; disambiguation, near-miss parsing and local learning work the same way. The network has a central representation of productions comprised of four pools of units (figure 10). The top pool has one unit for every nonterminal and represents that nonterminal as a parent. The bottom left (bottom right) pool also has one unit for every nonterminal and represents that nonterminal when it is the left-son (right-son). The terminal pool has one unit per terminal and represents that terminal as a son. To represent the production $A \rightarrow B\ C$, a production unit will have two-way excitatory links to the A unit in the top pool, the B unit in the bottom left pool, and the C unit in the bottom right pool.

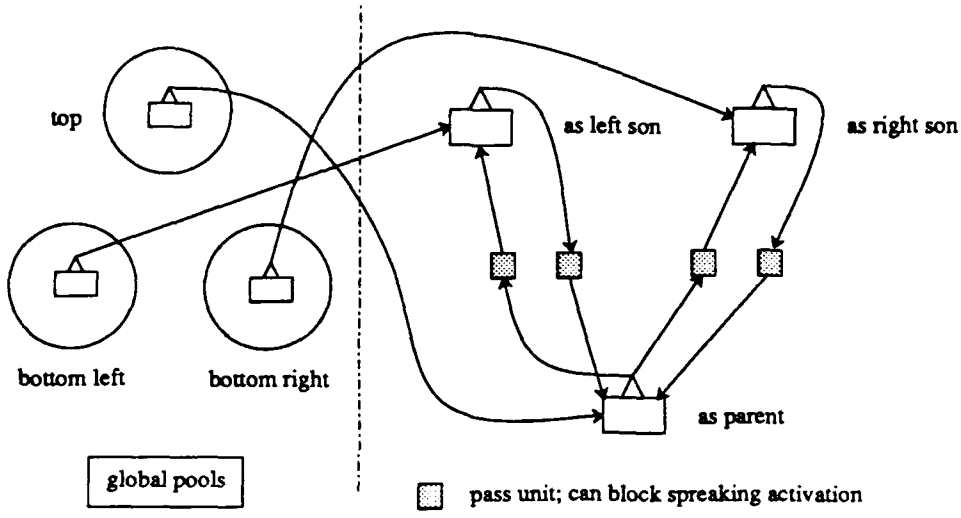

Figure 11: Nonterminal units in global learning network.

The global units are connected to nonterminal groups throughout the parsing network in the following way (see figure 11). Each unit in the top pool is connected to the top site of the parent unit in all the nonterminal groups for that unit. When a unit in the top pool turns on, the bottom units in all the corresponding nonterminal groups turn on. Each unit in the bottom left pool is connected to the bottom site of the left-son unit in all the nonterminal groups for that unit. When a unit in the bottom left pool turns on, the top left units in all the corresponding nonterminal groups become primed. There are similar connections from the bottom right pool to the right-son unit of each corresponding nonterminal group, and from the terminal pool to the terminal units.

For each connection described in the preceding paragraph, there is a reciprocal connection to the units in the central pools. If a unit in a nonterminal group becomes excited, it will cause the corresponding unit in the central pool to become primed. If exactly one unit in the top pool and one in each bottom or one in the terminal pool are primed, a unique production is represented by the pattern of activation. This is how the global production representation knows when local learning has occurred and what has been learned.

Global learning works as follows. The local learning mechanism causes some production to be learned. The parent and two children of the production instance learned become excited. In the parent nonterminal group, only the bottom unit becomes excited because the pass units in a nonterminal group do not pass on excitation. Similarly, in the left-son nonterminal group, only the upper left unit becomes excited, and likewise for the right-son. This causes the corresponding units in the central pools to become primed. (This does not seriously affect the parsing network.) The global production unit representing this production is now receiving activation from all three components and will become primed after two simulation steps. If more than one global production unit attempts to become primed at the same time, mutual inhibition will force them

all to zero potential and global learning of these productions will not occur. This prevents the cross talk which would occur if two productions were turned on in the central template simultaneously. Once a global production unit becomes primed, it inhibits all other global production units. The components of the production in the global nonterminal and terminal pools become inactive as soon as they stop receiving excited input from the network. The primed feedback from the global production unit is not enough to keep them active. Self feedback keeps the production unit primed. It will not turn on until the parse is finished. It requires excitation from the GlobalLearnUnit, which does not turn on until the network has finished a parse (during simulations, this unit is turned on by hand).

When the parse completes, the terminal units are turned off and the network calms down. The GlobalLearnUnit is turned on. This causes the still primed production unit to turn on, which turns on the global units comprising the production. For example, if the production to be learned is $A \rightarrow B\ C$, then the A unit in the top pool, the B unit in the lower left pool, and the C unit in the lower right pool will be on. This causes all bottom units in A nonterminal groups to turn on and all top left units in B nonterminal groups and all top right units in C nonterminal groups to become primed. Because the GlobalLearnUnit is inhibiting the pass units, activation will not spread within nonterminal groups.

The situation is just what is needed for local learning to occur all over the network. The bottom unit of the B nonterminal groups is on, but no fixed match unit is primed. For each division, there will be one free match unit receiving input from B as left son and C as right son. These productions will be learned. (This will not affect the global pools of units in their current state.) After a few steps, the global units become exhausted and turn off. They will remain quiescent for a few steps longer—enough to let the network calm down. When the network units lose input from these global units, they quickly die down.

In the network described so far global learning does not occur at the nonterminal group which learned the production locally. For example, suppose the local production instance A.4.3 $\rightarrow$ B.4.1 C.5.2 is learned. A little while later global learning takes place. B.4.2 and C.6.1 are primed and A.4.3 is on , but A.4.3 will not learn this production instance because B.4.1 and C.5.2 are also primed, which primes the now fixed match unit for this production, which inhibits the local learn unit for A.4.3. This is avoided by having the GlobalLearnUnit inhibit the inhibition of local learn units. It has a strong positive connection to all local learn units at the inhibit site. Recruitment of redundant match units is prevented by having fixed match units inhibit free match units with the same division. If the production being learned is already known for some division, then that fixed match unit will become primed, and no new match unit will be fixed. This is an extension of a mechanism already in place: free match units already inhibit other free match units (ranked lower) within the same division. These inhibiting connections remain even after the unit becomes fixed. All we need to add are inhibiting connections from match units which begin life fixed. With this addition, the new production can truly be learned globally. It also makes the network more robust, as described in the next section.

The network builder was programmed to make two unfixed match units per division per nonterminal group in order to test that no redundant productions were learned.

Simulations for the following grammar worked correctly (the results are somewhat big for a table, so they will be summarized):

$$S \rightarrow A\ B \quad A \rightarrow a \quad C \rightarrow c$$
$$B \rightarrow B\ C \quad B \rightarrow b$$

First, the network was run with the input *b c a b*. This resulted in a near-miss parse (actually the miss was not too near). The production instance A.1.3 → B.1.2 A.3.1 was learned. This caused the three units A.1.3.parent, B.1.2.lson and A.3.1.rson to become excited, which primed the global units A.parent, B.lson and A.rson, which primed A>B.A, the global production unit for *A → B C*. A couple of steps later the excited network units calmed down to on. This caused all the global units except A>B.A to turn off. When the parse completed, the terminal units were turned off and the network was run until only A>B.A was left primed (about 12 steps). The GlobalLearnUnit was turned on by hand, which turned on A>B.A, which turned on A.parent, B.lson and A.rson. On input from these units turned on all A.x.y.parent units and primed all B.x.y.lson and A.x.y.rson units. Learn units for all the A.x.y groups came on. The ones with length greater than one excited free match units, one per division, which learned their new productions.

The A.1.3 group had its fixed match unit for A.1.3 → B.1.2 A.3.1 primed because of the production instance it had just learned. This did not prevent the learn unit from coming on because the GlobalLearnUnit was on. It did prevent the other free match unit with the division 2 + 1 from activating. One free match unit for the division 1 + 2 became excited and learned the production instance A.1.3 → B.1.1 A.2.2. After a few steps, the global nonterminal units (but not the GlobalLearnUnit) became exhausted as did all the learn units. The network quickly calmed down.

After learning, the network was tested with the input b b a a b, which requires the use of two new instances of the production A → B A. The parse completed successfully. The following units were on when the network stabilized:

S.1.5.par					
A.1.4.lson					
A.1.4.par					
	A.2.3.rson				
	A.2.3.par				
	B.2.2.lson				
	B.2.2.par				
B.1.1.lson	B.2.1.lson	C.3.1.rson	A.4.1.rson	B.5.1.rson	
B.1.1.par	B.2.1.par	C.3.1.par	A.4.1.par	B.5.1.par	
b.1	b.2	a.3	a.4	b.5	$.6

2.5.3 Deferred Learning

The global learning described in the previous section would fail to distribute productions learned locally if more than one is learned during a parse. It is possible for the network to take advantage of periods of quiet to catch up on unlearned productions. The mechanism I propose (but have not implemented) is spontaneous excited activity of fixed match units when the network is quiet. The probability of a match unit becoming excited would be inversely proportional to the length of time it has been fixed, so that recently fixed match units would be more likely to become excited. The excited match unit would excite the parent and children of the production, which would start global learning as in the previous section. More than one match unit activating simultaneously would do no harm. Global learning of already learned productions will have no effect as described above.

One way to accomplish this is to run a link from the global learn unit (which could just as easily be a network of units) to match units. The link would connect to a new site. The global learn unit is on when parsing is not taking place, so if the match unit responds to input to this site by becoming excited probabilistically, we achieve the desired effect. The weight on this link could be made non-zero when the unit becomes fixed and gradually decay after that. Recently-fixed match units would get the most activation and, thus be more likely to fire.

Rehearsing productions also makes the network more robust. If a match unit were to "die," then another would be recruited to take its place the next time its production is rehearsed.

2.6 Discussion

The major advantage of the parser is its generality and its quick, sure results. By maintaining all possible parses in parallel, the need for search (i.e., relaxation) is eliminated. One of the goals of connectionist models is to account for the solution of complex tasks in a few computational steps using massive parallelism. This network does exactly that.

The major disadvantage of the parser is its rigid structure and fixed length. Because the length of the network is fixed, the set of strings it can parse is finite. It is in fact a finite state machine.[3] Any physically realized context-free parser has finite limitations, in the size of parse tree which can be represented if nowhere else, but one might hope for an extendible structure or graceful degradation. It would be a major improvement if the network could parse longer strings by acquiring more resources (i.e., units) on the fly. McClelland's [1985] CID mechanism may prove useful in this capacity [McClelland and Kawamoto, 1986], though the resource requirements of CID are substantial.

The practice of processing a sequence of inputs using a spatially replicated mechanism has inherent difficulties. While it may prove useful on some scale, there is certainly a point at which it breaks down. No story understander could absorb an entire story in parallel, to take an extreme example. Replication of resources is wasteful and is compounded when the network is integrated into other mechanisms, such as the semantic processing of the input. It also makes learning difficult.

[3]Thought of as such, it has an efficient distributed representation of the states where the pattern representing each state is isomorphic to (partial) parse tree(s) for the string.

The learning mechanism described is certainly inadequate as a theory of language acquisition, but in keeping with the purpose of the thesis, it does demonstrate some techniques which may prove useful. If one must use multiple copies of subnetworks which perform the same function, learning in such a way that all copies are kept consistent is necessary.

The parsing algorithm upon which this work is based is an example of dynamic programming [Aho *et al.*, 1974]. Other problems with efficient dynamic programming solutions may have similar parallel implementations. One such problem is finding the number of edit operations (insert, replace and delete) required to convert one string to another. A fast implementation of this algorithm might be useful in tasks requiring pattern matching. The values passed during a computation are the number of edit steps so far. This means that a potentially large amount of information must be communicated, unlike the parsing network in which only the existence of constituents is communicated. While this does not stand in the way of a fast parallel implementation, it may present complications for a connectionist model, where the output values of units are not meant to carry much information [Feldman, 1982].

Chapter 3

SIMPLE CONCEPT LEARNING

3.1 Introduction

This chapter presents the results of experiments on feature-based concept formation. A "feature-based" concept is one without internal structure; it can be represented as a list of attribute values. This is the most common task encountered in the learning literature. Learning structured concepts—those with subparts and relations between the subparts—is more difficult and is the subject of chapter 4. The use of connectionist networks is an *a priori assumption of this work*. The goal is not only to develop the best learning techniques possible, but to advance our knowledge of connectionist methodologies. The chapter begins with a general discussion of the problem, then gives a brief summary of some relevant previous work. An original concept learning technique using pair recruitment is described in detail; experimental runs are reported and discussed. The technique works well in some circumstances, but has some serious flaws as well. A second concept learning technique is developed. It is a simple modification of competitive learning to make it responsive to reinforcement. The chapter concludes with experiments on generalization, back propagation and structure.

3.2 Goals and Issues

3.2.1 Representation of input and output

In the experiments described in this thesis, the inputs are represented by selecting a single value for each of a small number of attributes. The value sets are discrete and fixed, so there are only finitely many possible inputs. For example, there may be four choices for color: red, blue, green, and yellow; and three for size: small, medium, and large. There is also a small number of fixed concepts or category labels. In the simple case, only one concept per input will be chosen. More generally, more than one or none may apply. In this case, it is more natural to refer to the concepts as properties.

The input representation is easy to work with but is too limiting for many natural attributes, such as those which lie on a continuum, e.g. size and weight. Perhaps

more important, it does not allow the model to generalize over feature values (see the discussion in Holland, et al. [1986]). Orange is no nearer red than blue is. Clearly this is not correct, but was used in order to focus on the recruitment of hidden units. Representing continuous values of features in a connectionist network presents various difficulties [Ballard, 1986]. These problems are not dealt with here, though one must be careful not to rely on simplifying assumptions which are not valid in the general case.

The number of categories each input may belong to affects the task considerably. If the categories are mutually exclusive, they can be mutually inhibitory. This means that positive feedback to category A is implicit negative feedback to category B. This seems a natural way to provide negative feedback. If, however, more than one category may apply, it may be necessary to explicitly provide negative feedback. This is somewhat less intuitive. When learning, we tend to find out what something is, not what it's not. In reality, some concepts are mutually exclusive while others are not. This could even be learned. Concepts which are mutually exclusive will be so represented; properties which co-occur will excite each other. The experiments reported in this thesis assume mutually exclusive categories.

3.2.2 Hidden units

Connectionist networks have been applied to feature-based concept learning tasks for decades. When the concept definitions are linearly separable, there is a learning algorithm which is guaranteed to find a workable set of weights after sufficient training. In general, however, at least one layer of "hidden" units between the input and output is required [Minsky and Papert, 1969]. Finding an algorithm to learn the weights on links to these hidden units has occupied connectionists for some years and is the focus of this chapter. There are exponentially many combinations of input features. Only those important to the task should be explicitly represented. The first thing to be resolved is the nature of the hidden unit's support.[1] At one extreme, there could be no constraints. So there might be a hidden unit which activates proportional to $0.4red - 1.3large + 1.9yellow + \cdots$ At the other extreme, the hidden units could be constrained to represent, e.g., only pairs of features, each weighted equally. My intuition suggests that the restricted approach will yield quicker solutions, be more easily understood, and more likely to prove useful in future tasks, *if* the restriction is chosen correctly. This is certainly true in abstract tasks where the form of definitions is constrained. It may be less true for natural concepts, which may prove harder to constrain a priori. Restricting the support of hidden units is a major motivation for the work described in section 3.4.

3.2.3 What is to be learned

Before generating training examples, we must choose some definition of the concepts. Conjunctive definitions are the simplest possible. An object belongs to a conjunctive class if and only if all the properties of that class apply. Conjunctive definitions are easy to learn without hidden units. Disjunctive definitions complicate matters considerably (for a theoretical discussion, see [Kearns *et al.*, 1987]). Any boolean combination of

[1] The set of units with strong connections to this unit.

feature values can be expressed as a disjunct of conjuncts, e.g (small and red) or (square and yellow); this is the form of disjunctive definition which will be used.

Such definitions are often used in artificial concept learning tasks, but may be a poor model for natural concepts. Probabilistic definitions [Smith and Medin, 1981] may be more appropriate. These might involve specifying a prototype for each concept. Membership would be defined by some distance metric and could be either binary (all-or-none, depending on the distance) or gradient. The definitions used in the experiments reported below were disjunctive and nonprobabilistic, though training with noisy data did approximate probabilistic concepts. Independent of noise, the networks did respond in a graded manner to the definitions.

3.2.4 Nature of the training

The specificity of the feedback can vary from exact to none at all. The three major classifications are:

- Supervised learning. The desired answer is given in its entirety [Rumelhart *et al.*, 1986].
- Reinforcement learning. Only the correctness of the answer is given [Barto, 1985].
- Unsupervised learning. No feedback is given [Rumelhart and Zipser, 1985].

The experiments reported below used supervised learning, though certain aspects of reinforcement and unsupervised learning do come into play.

The training can be made imperfect to simulate more realistic tasks where input errors are probable. Connectionist networks are generally robust, a result of making small adjustments for each input instead of making a discrete change of hypothesis. It is important that new connectionist learning techniques retain this property. Error can be added to an input/concept pair either by changing the input (measurement error) or the concept (classification error) or both [Cohen and Feigenbaum, 1982]. These can have different effects on performance. A measurement error may present unusual combinations of input features, which may be "misclassified." If this causes the recruitment of a representation of those input features, resources will be wasted. Measurement errors are used in experiments described in this chapter.

3.2.5 Computational limitations

First, a connectionist network (see chapter 1) must be used. All learning must be incremental: after each input is presented, the network changes and the input is discarded. This precludes gathering statistics about the whole body of training data and building some kind of optimal decision strategy. This is a sensible restriction, as the eventual goal is to build networks which will adapt, in real time, to very complex environments with a open ended sequence of inputs. There is a price to be paid for using incremental learning. The resulting network will probably not be optimal, and more data presentations will probably be necessary. The advantages of an incremental algorithm are less clear if the data set must be cycled through hundreds or thousands of times (see section 3.2.6). Various forms of compromise are possible. For example, remembering

some fixed number of past trials is feasible, though none of the networks described in this thesis did so.

The amount of memory cannot be unreasonably large. For example, representing all possible combinations of features [Hayes-Roth and Hayes-Roth, 1977] is not possible. I have set no hard limit for the number of hidden units allowed, though at most a few times the number of necessary hidden units were used. Equally important, though often ignored, are limitations on the connectivity patterns. It may be feasible to totally connect layers of units in small networks, but this will not scale to massive "real-world" networks. The connectivity problem was not uniformly addressed in the networks described below and will be discussed later.

3.2.6 Speed of learning

The goal is to have the network learn in roughly the same number of trials as human subjects on abstract learning tasks—those involving no world knowledge. For example, the subject may see figures which vary on four dimensions, e.g., size, shape, color and border-width. The concept to be learned may be "large and red," in which case just a few examples should be enough. Disjunctive definitions are more difficult [Brunner *et al.*, 1956] and may take considerably longer to learn. The idea is not to model the number of trials in any exact way, but to be approximately the same.

3.2.7 Meaningfulness of the weights and response functions

The basic question is: should the weights have some independent interpretation, such as probability or log-likelihood? If the answer is "yes," there are two problems. First, a connectionist weight-change rule must be found to maintain such weights. Second, the unit response function must correctly, or at least intelligently, interpret these weights. Also, if the nature of representation is predetermined, then the network will be less likely to discover unforeseen and useful representations.

On the other hand, if the weight change is guided by error correction alone, where the magnitude of the weight change is a function of the error, then the representation used may be hard to understand: a weight of -1.3 does not mean anything in particular, except that in conjunction with the other weights, it works. Also, if the weight change is purely error-driven the network may not learn non-essential correlations: if the behavior is correct, no change will be made. Nonessential information might be useful. The weight-change rules used in this thesis represent a combination of these approaches.

3.2.8 Resource utilization

Since the recruitment of hidden units will be based upon imperfect evidence, some will inevitably be recruited unnecessarily. In order to save resources, it is highly desirable to reuse or adjust hidden units which are not being productively used. The difficulty lies in determining when a hidden unit is useless. Many connectionist learning algorithms rely on repeated, interleaved presentations of data from all classes to be learned. Representations which are not periodically reinforced will be lost. It is not clear how well this will work for more realistic tasks, which may have essential functions not frequently

encountered. The pair-recruitment technique described below never reuses resources, which is clearly unsatisfactory as well.

3.2.9 Other considerations

The most important requirement of a concept learning network, besides competent performance, is its ability to be integrated in a larger framework. This is especially important for connectionist networks, as they do not lend themselves easily to modular design. The outputs of step one cannot be sent to step two in a list. Ease of integration has not offered much guidance as the framework into which everything must be integrated is not yet known. Still, certain principles can be deduced. For one thing, there should be no crucial timing assumptions of the nature "run 10 steps, then provide the feedback and run 10 more steps." It is hard to predict what further computations will be based upon the activation of the concept units or how long they will be required to be on. A weight-change rule which increments the weight every simulation step for which the sending and receiving units are active might break down in a more complex context. The network should be robust over timing variations. Chapter 4 hypothesizes the need for a sequential component in the processing of structured objects. The changing focus of attention at the input level precludes the use of many learning algorithms which rely on the input staying fixed for the entire trial.

The learning algorithm must scale. Statistical properties of large networks can be qualitatively different from those of small networks. Since only small networks can be efficiently simulated, most of the work will be done with small networks. One must not succumb to the temptation to optimize the performance of small test networks in a way that will not work on larger, more realistic networks.

While the primary goal of the network is to categorize inputs, a secondary goal is to instantiate concepts, i.e., activate features given concept activation. This entails learning correct weights on links from concepts to hidden units and features, and from hidden units to features. Learning first-order correlations between concepts and features is fairly straight-forward. Keeping disjunctive definitions straight is more difficult. Suppose our concept is defined as small and furry or large and feathered. What we don't want to do is instantiate small and feathered. The first learning network described cannot instantiate conjunctive concepts consistently; the second can.

3.3 Previous Work

There is a long (for computer science) history of connectionist learning algorithms. Good overviews have appeared recently [Hinton, 1987, Lippmann, 1987]. This section briefly reviews work related to the thesis. Rosenblatt's [1962] perceptrons are linear threshold devices. They take the weighted sum of their inputs and respond positively if the sum is greater than their threshold. Perceptrons change their weights in response to reinforcement using a simple and intuitive weight change rule: if the unit's response is positive and should be negative, then decrease the weights on links from positive units and increase the weights on links from negative units; an opposite change is made if the response is negative and should be positive. The perceptron learning algorithm will provably converge to a solution if the problem is linearly separable, i.e., if there is

a hyperplane in the multidimensional space defined by the perceptron's weight vector such that the positive and negative instances are separated by that hyperplane.

The assumption of linear separability is key. Minsky and Papert [1969] showed that there are many simple problems for which no perceptron solution exists. The simplest of these is the exclusive-or mapping from two inputs to a single output such that the output's response is positive just when exactly one of the inputs is positive. Note that Minsky and Papert proved not just that there is no guarantee of convergence on the solution, but that no solution exists. In order to solve exclusive-or and many other problems, high-order combinations of inputs must be taken into account. Extra units between the input and output are needed. These units are often called "hidden" because they receive neither input nor feedback directly from the external environment.

A great deal of effort has been expended on weight-change algorithms for hidden units. Barto's [1985] A_{RP} algorithm uses a single global error evaluation which is broadcast to the hidden units, combined with a stochastic search through the weight space to find a solution. The search can take a long time, and does not promise to scale well; a purely stochastic search through a space of thousands or millions of weights in search of solutions to complex problems is not feasible.

A recent major advance in hidden unit learning [Parker, 1985, Rumelhart *et al.*, 1986] derives a gradient descent rule which propagates exact error information from the output units to the hidden units based on minimizing the sum of error squared and the chain rule. These back propagation networks consist of layers of feed-forward units. In the variation described in [Rumelhart *et al.*, 1986], each unit takes a weighted sum of its inputs and responds according to the *logistic* activation function:

$$o_j = \frac{1}{1 + e^{-(\sum_i w_{ji} o_i + \theta_j)}}$$

where o_j is the output value of unit j, w_{ji} is the weight on the link from unit i to unit j and θ_j is the bias of unit j. The error signal for an output unit j is

$$\delta_j = (t_j - o_j)\, o_j (1 - o_j)$$

Here, t_j is the desired output. The error signal for a hidden unit j is

$$\delta_j = (1 - o_j)\, o_j \sum_k \delta_k w_{kj}$$

The summation is the back propagation of error from higher level units. The weight change rule is

$$\Delta w_{ji}(n+1) = \eta(\delta_{pj} o_{pi}) + \alpha \Delta w_{ji}(n)$$

Where η is the learn rate and $\alpha \Delta w_{ji}(n)$ is a heuristic, momentum term which greatly improves the performance of the search.

This technique has been quite successful. However, there are some problems with it:

- It is often slow. (Though I have found it capable of rapid learning when the task is simple, especially when there are direct connections from the input to the output.)

- It does not scale well. Other researchers [Plaut, 1986] have noted that back-propagation does not scale to large networks easily.
- It only responds to training. In other words, it does not continue to change after it learns to correctly classify the input. Continued learning is desirable for less important features for which typicality information may prove useful even if these features are not crucial to category decisions.
- It unlearns too readily [Sutton, 1986]. Unless there is frequent retraining, hidden units used to encode a previously learned task will be used for a new task, even if there are other, uncommitted units available.
- It uses a special mechanism for back-propagating link-specific errors. "Pure" connectionist implementations of this are technically possible, but awkward. Rather precise control is required to manage the two separate stages: forward summing of evidence and backward propagation of error and weight-change. Hidden units must wait until all errors from higher layers have been back-propagated before making any changes. When links cross more than one layer, this requires additional control mechanisms.
- It does not easily allow cycles in the link connections. This precludes instantiating features by activating concepts. It also eliminates mutual inhibition or support within a layer, e.g., among the concept units. Mutual inhibition and support are important tools in connectionist computation.
- It does not do temporal credit assignment. Feedback must be given for the immediately preceding input.

While back propagation is an important advance in connectionist learning, there is still *motivation to explore new techniques.*

Competitive learning [Grossberg, 1976b, Grossberg, 1976a, Rumelhart and Zipser, 1985] is an unsupervised learning technique in which the input units are connected to one or more clusters of competitive units. Within each cluster, the competitive units are mutually inhibitory, so that only one will remain active for each input. The total sum of weights to each competitive unit is held constant at 1.0, with the initial configuration being random. When an input pattern is presented the winning unit changes its weights so as to be more likely to respond to a similar input in the future: links from active input units are increased; those from inactive units are decreased. Simple regularities, based on the number of shared input units, will be discovered by the competitive units which will partition the input space accordingly.

A great deal of literature exists within symbolic AI on learning [Cohen and Feigenbaum, 1982, Michalski *et al.*, 1983, Michalski *et al.*, 1986]. That which is most relevant to the work presented here is Mitchell's [1985] candidate elimination algorithm. His search for a concept description is analogous to the connectionist search for hidden units. The number of possible combinations of input features is exponentially large. Mitchell narrows in on the correct concept description by maintaining the boundaries around the possible hypotheses in the state space. Positive examples of the concept make the specific boundaries more general and negative examples make the general

boundaries more specific. If all goes well, a single, correct hypothesis will eventually be isolated. Unfortunately, this algorithm cannot easily handle disjunctive definitions. The problem is determining whether a counter-example should cause a generalization of an existing description (which one?) or begin a new description. Also, the algorithm is not noise tolerant. Each example forever trims away part of the hypothesis space. An error can cause the correct answer to be irretrievably thrown away.

Recent work by Schlimmer and Granger [Schlimmer and Granger, 1986, Schlimmer, 1987] is similar to the pair-recruitment technique presented in this chapter. Their model is not connectionist, but it does classify inputs based upon weighted combinations of features. Like Mitchell, they search the hypothesis space—feature nodes which represent conjunctions and disjunctions of features. Instead of trying to find a single description of the concept, they search for a set of feature nodes which is sufficient to distinguish the concepts in question. These nodes correspond to the hidden units needed for connectionist networks. The algorithm is noise tolerant because feature nodes are not eliminated due to a single input. The evidence each input provides for a concept is calculated according to exact Bayesian formulae. Logical sufficiency (LS) approximates the degree to which the presence of a feature (F) increases expectation of an outcome (O):

$$LS = \frac{p(F \mid O)}{p(F \mid \neg O)}$$

Logical necessity (LN) approximates the degree to which the absence of a feature decreases expectation of an outcome:

$$LN = \frac{p(\neg F \mid O)}{p(\neg F \mid \neg O)}$$

The expectation of class membership for some instance is computed by projection:

$$\text{Odds}(\text{positive} \mid \text{instance}) = \text{Odds}(\text{positive}) \times \prod_{\forall \text{matched}} LS \times \prod_{\forall \text{unmatched}} LN$$

The suitability of these formulae for connectionist networks is questionable. It is not possible to maintain accurate probability estimates using weight change rules; the multiplication in the calculation of the odds would exaggerate any error, even for apparently insignificant features. The major obstacle to a connectionist implementation of this system is the complex set of heuristics used to form new feature nodes.

3.4 Pair Recruitment

This section describes a connectionist network which learns to categorize input. The category definitions used are Boolean combinations of input features. No negation is used, in order to focus on the recruitment of important conjunctions of input features. The goal is to learn the definitions in as few trials as possible. In order to achieve this, a learning algorithm optimized for the task is used instead of a general-purpose algorithm. The belief is that the task is general enough for such an algorithm to be useful. The hidden units which are necessary to solve the task represent conjunctions of key features. Defining features should co-occur more often than other features The

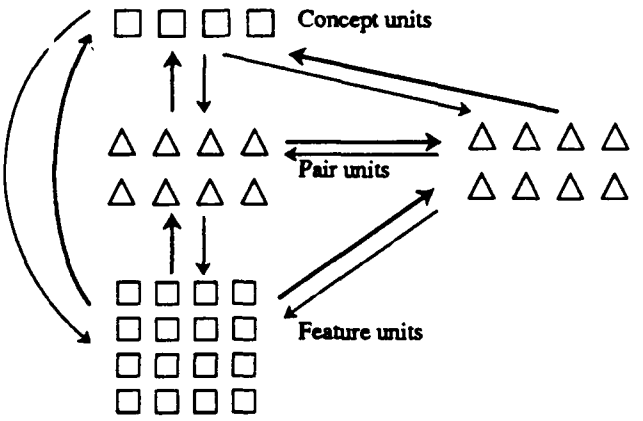

Figure 12: Layout of the network.

recruitment of hidden units is based on frequency of co-occurrence of inputs. In order to further isolate input features which need a high-level representation, *surprise* is used to enable recruitment. Surprise occurs when the network's classification does not match its feedback. Conjunctions of feature which need high-level representation will co-occur during surprise until the recruitment is successful.

The weight change rule used on links to concept and feature units keeps the weight at approximate the probability of post-link activity given pre-link activity, e.g., the probability that an object is a crow given that it is small and black. The weight change rule used on links to hidden units is designed to isolate two strong connections from a greater number of initial weak connections. Once a hidden unit has been recruited to represent some pair, the weights become stable. The hidden units represent pairs rather than arbitrary conjunctions because pair recruitment is easy to implement. Longer conjunctions can be achieved through a hierarchy of pair units (though this extension proved troublesome).

3.4.1 Network structure

Figure 12 shows the layout of the network. Figure 13 shows the structure of the units representing the values for the attribute color in detail. The color units are mutually inhibitory,[2] enforcing the requirement that only one be active at a time. An input is presented to the network by activating external units. These units activate the corresponding attribute-value units. The attribute-value units could be designed to stay on when set, eliminating the need for separate external units, but having separate, external activation seems more natural. It also lets units detect differences between external input and internal feedback.

The use of distinct external units is especially important for the concept units, which have the same structure as the value units for one attribute. They are mutually inhibitory and can receive input from external sources. This input serves as the teacher in the network. Discrepancy between the teaching input and the internal evidence causes network surprise and learning.

[2]Inhibitory links are indicated by a circle on the end of the arc.

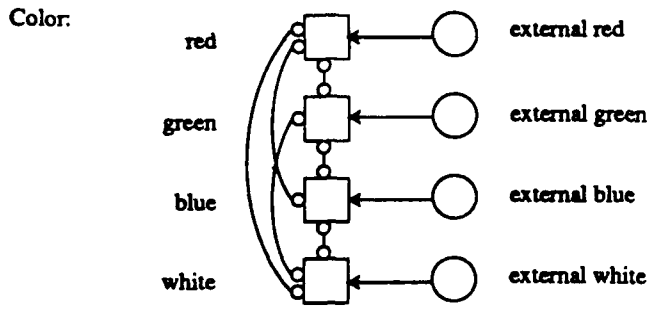

Figure 13: Representation of four values for the attribute color.

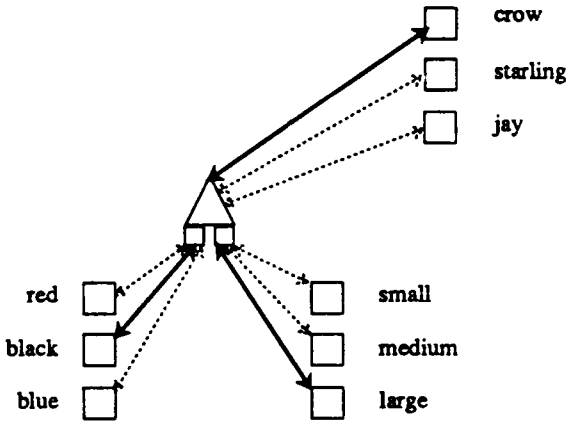

Figure 14: Hidden unit representing *large* $\wedge$ *black* is evidence for *crow*.

All attribute-value units are connected to all concept units and vice-versa. These links represent the evidence each provides for the other. The links to the concept units are more important for this work. They are used to categorize the input. The links from the concept units to the attribute-value units are used to set default values for concepts.

Pair units start life with weak two-way connections from many attribute-value units and to all concept units. The pair units have two sites for bottom-up activation from the attribute-value units. By isolating one link to each site, the pair unit learns to respond to some pair of inputs. Figure 14 shows the connections of a pair unit after it has been recruited. The set of pairs which the pair unit could possibly represent is the cross product of the links to each site. The connectivity of each pair unit is chosen randomly with fixed density. Care is taken to insure the same unit is not connected to both sites to prevent pairs of the same unit. Section 3.4.3 discusses the effects of varying the density of connections to the sites. Some of the simulations use higher-level pair units which connect to the first layer pair units and to the attribute-value units. These units represent triples of attribute values.

It is necessary to communicate a concept unit's surprise (external feedback disagrees with internal evidence) to the pair units. Enabling learning for whole populations of

units seems natural. Without some such mechanism, learning would occur at an equal rate all the time, which is not desirable. The implementation of this broadcast is somewhat ad hoc, but it works. When a pair unit is surprised, it becomes hyperactive. This activates a special learn unit, which is connected to all the pair units. Excitation from the learn unit is required before the pair units will learn. Section 3.4.3 describes how this learn-enabling network can be used to incrementally enable the pair units.

3.4.2 Network behavior

A typical learning trial proceeds as follows. External input units instantiating some concept are activated, which activates attribute-value units. Pairs of these activate recruited pair units. Together, they activate concept units. The concept unit with the strongest input remains active; the others become quiescent. When the external unit corresponding to the correct concept is activated externally, it provides very strong input to the corresponding concept unit. If the correct concept unit is on weakly or not at all, it enters a surprised state, which accelerates learning at that unit. A surprised concept unit becomes hyperactive, which actives the special learn unit, which enables the pair units.

The first three subsections below describe the response of the units to their input. The next two subsections detail the rules for evidential and structural (pair recruiting) weight change.

Concept units

The concept units distinguish three sources of input:

1. Excitatory links from attribute-value and pair units.
2. Inhibitory links from other concept units.
3. The external (teaching) link.

If the external input is strong, the unit always takes that value. In the absence of external input, if the inhibition is stronger than the excitation, the unit becomes inactive. Finally, if the excitation predominates, the unit's potential becomes:

$$.4 \times \text{old-potential} + .6 \times \text{magnitude-of-excitation}$$

Putting the old potential in the formula prevents oscillatory behavior (the simulations are synchronous).

The magnitude of the inhibition is taken to be the maximum of all inhibitory inputs. This realizes the desired behavior, as a concept unit needs to know whether it is the strongest. The magnitude of the excitation is also taken to be the maximum of the excitatory inputs. If a hidden unit representing the concept definition has been recruited, using the maximum is ideal. However, if this is not the case, then evidence for the concept can only be measured as a percentage of expected strong inputs (components) which are actually present. Experiments using a sum of excitatory inputs instead of max were largely unsuccessful. The problem is that many inputs with small weights will have the same effect as one strong input. Each concept unit has a link from all

the attribute-value and pair units. The weights on these links have a small random fluctuation; they are rarely zero. This problem is exacerbated by the ceiling on the weights, which are mock probabilities. If the noise is in the range [0.0, 0.1], then ten noisy inputs can equal one strong input in sum.

While the maximum rule works well in some cases, it has disadvantages. For example, suppose concept C1 has a single strong input of magnitude .9 and that concept C2 has 10 strong inputs of magnitude .89. It is not at all clear that concept C1 should win. More important, if an exact representation of the concept definition does not exist, then it is only by the presence of many subconstituents that the concept can be recognized. For example, if we define a crow to be something which is large, black and flies, but have no "large $\wedge$ black $\wedge$ flies" unit, then the max rule is in trouble. The maximum evidence from the "large", "black", "large $\wedge$ black", etc. units may not be that impressive, but the combination is. This property is, in fact, central to most connectionist models [Rumelhart and McClelland, 1986, McClelland and Rumelhart, 1986].

Pair units

A pair unit has two sites which receive weighted inputs. Each site computes a value which is equal to its maximum weighted input. The unit's potential is equal to the minimum site value. The idea is that the unit responds only to a specific pairs of inputs. One member of the pair has a strong connection to the left site; the other to the right site. Both must be active before the pair unit responds.

Pair units also receive top-down input from concept units at another site. This link represents the likelihood of the pair given the concept. Because top-down feedback should be less salient than direct "perception," this input is only half as excitatory as bottom-up input. Also, positive feedback loops would be a problem if activation were equally strong in both directions.

Attribute-value units

The attribute-value units distinguish three sources of input:

1. Excitatory links from concept and pair units.

2. Inhibitory links from other attribute-value units.

3. The external (input) link.

The maximum weighted input from each source is computed. As with concept units, the external input overrides all others and will force the corresponding attribute-value unit on. In the absence of external input, the units' response is equal to the maximum excitation if it is greater than the maximum inhibition and is zero if the inhibition is greater. The top-down excitatory inputs are at most half as powerful as the external inputs, so external excitation will always overpower internal feedback.

Evidential learning

The evidential weight from unit A to unit B reflects the probability of unit B being active given that unit A is active. The weight change rule which maintains, given certain assumptions, the weight from A to B at approximately $\mathcal{P}(\mathrm{B} \mid \mathrm{A})$ is

$$\Delta w_{t+1} = \begin{cases} \epsilon(1 - w_t) & \text{if A and B are active} \\ -\epsilon w_t & \text{if A is active and B is not} \\ 0 & \text{if A is not active} \end{cases} \quad (1)$$

This rule was derived by assuming that the current weight represents the correct probability. If the weight is 0.8, then B is expected to follow A four times as often as not. Therefore, to keep the weight stable, the reward for following is one-fourth the punishment for not following. As a weight wanders away from its "true" value, it will be pushed back. For example, if the weight should be 0.8, but is currently 0.6, its reward will be bigger than it was at 0.8. The weight does not change when A is not active.

This rule favors recent experience, and will perform poorly on uneven distributions. If 80 positive trials are followed by 20 negative ones, the weight will only coincidentally be near 0.8 at any time.

The normal ϵ for bottom-up weight change is .01. However, when the concept units are surprised, the value of ϵ is increased to .1. Surprise occurs when the external input is significantly larger than the internal evidence. An analogous *suppressed* state occurs when a strongly active concept unit is suddenly inhibited by another concept unit. This state also accelerates learning. These two states add a kind of error-driven learning to the probabilistic learning.

Structural learning

Pair units begin with several connections to each bottom-up site. The weights are small; the pair unit cannot become more than slightly active. The rules for learning are the following. First, no learning happens unless the unit is in a *learn* state, which is caused by surprise at the concept level. Second, a link's weight change is proportional to the pair unit's activity and the weighted input of the link. In addition, the unit has a floating threshold. When it receives strong input, it increases its expectation; even stronger input will be required to activate it in the future. This serves to isolate the strong connections.

Eventually a pair unit learns to respond to a single pair of inputs. Suppose it enters the *learn* state and is somewhat active. This causes a slight increment in the weights from all active inputs. Later, when some of these same inputs activate the unit and it is in the *learn* state, they will be incremented even more because (1) the unit will be more active, and (2) their own weighted contribution is greater. An analogy can be made with a race in which the the length of a runner's stride increases proportional to his lead. Eventually, some of the inputs will become strong enough to maximally activate the unit. When this happens, the threshold of the unit increases, so that stronger input will be required to activate it in the future. Weak weights which could have caused some learning originally now have no effect. Race conditions will tend to limit each site to one strong input. An additional force has been added to insure this; active pair units decrease their inactive links. This makes more than one strong input to a site unstable.

3.4.3 Fine tuning pair units

Analysis of connection density

The pattern of connections from attribute-value units to pair units is important. The connections are random, with redundant links removed. If the density of connections is too high, especially important pairs will tend to recruit too many pair units. If the density is too low, some pairs may not be represented enough.

The following probabilities help determine what the density of connections should be. Let

v = number of attribute-value units
d = number of connections to each site of a pair unit
c = number of pair units

Redundant connections to a single unit are avoided, thus d is at most $v/2$. Then, R = the probability a particular pair is representable at a single pair unit is (derived using a counting argument)

$$R = \frac{2d^2}{v(v-1)}$$

Given d inputs to the one site, the other site has only $v - d$ possible inputs because there are no duplicate connections. The probability some pair will be totally excluded is

$$X = (1 - R)^c$$

The expected number of pair units which could represent each pair is

$$E \simeq Rc$$

The following formula approximates the dangers of a too-dense connection pattern. Suppose certain pairs are so persistently present when the network is surprised that they recruit every pair unit which they could recruit. What is the probability that some other pair will still be represented at a free pair unit? Let s be the number of dominating pairs we wish to escape. The probability of avoiding one such pair at a single unit is $(1 - R)$. The probability of avoiding s other pair units lis approximated by $(1 - R)^s$. (Because the pairs are not independent, this is not accurate.) There are E expected units at which a pair has a chance. The probability of avoiding dominance by s pairs at one of these is approximately

$$\text{probability of escaping dominance} = 1 - [1 - (1 - R)^s]^E$$

Table 6 shows some sample values, with the number of attribute-value units, v, held constant at 16, and the number of pair units, c, held constant at 20. These values have been used frequently in simulations. The density of connections to each site varies as shown.

The probabilities are somewhat better with large numbers of units. For example, with 5000 input units, 100,000 pair units and a connection density of 40 links per pair unit site, the probability of exclusion is .001. The probability of escaping total recruitment by the top 20,000 pairs is .87; by the top 50,000 pairs is .23.

Table 6: Some probabilities involving pair recruitment.

density	prob at each	prob excluded	expected number	prob of escaping dominance by 1	2	3	4	5
1	0.008	0.846	0.167	0.550	0.495	0.460	0.434	0.413
2	0.033	0.508	0.667	0.896	0.837	0.789	0.748	0.710
3	0.075	0.210	1.500	0.979	0.945	0.905	0.861	0.817
4	0.133	0.057	2.667	0.995	0.975	0.940	0.891	0.833
5	0.208	0.009	4.167	0.999	0.984	0.943	0.875	0.788
6	0.300	0.001	6.000	0.999	0.982	0.920	0.807	0.668
7	0.408	0.000	8.167	0.999	0.970	0.850	0.656	0.459
8	0.533	0.000	10.667	0.999	0.927	0.681	0.404	0.212

Notice in table 6 that in order to have a satisfactory probability of not excluding some pair, the density must be at least 5 and preferably 6. But this means that the probability of being shut out by 5 stronger pairs is unacceptably high (0.2).

Lateral inhibition of learning

As suggested by the above analysis, some important input pairs were not recruited in simulations. The first fix attempted was to increase the density and discourage over-representation by having pair units inhibit the *learning* of their neighbors. They did not inhibit the activation of their neighbors. This is important. To prevent the recruitment of too many large-and-black units, the network should not allow learning to take place simultaneously at too many pair units. However, neighboring pair units which represent different pairs should be allowed to be simultaneously active when both those pairs are present in the input.

This lateral inhibition does not improve performance. The problem occurs when long definitions are used. There are a lot of pairs in a long definition which *should* be recruited simultaneously.

Incremental enable

A bigger performance improvement results from incrementally enabling the pair units. The idea is to allow only a fraction of the pair units to participate in recruiting at first. As those units begin to learn pairs, more pair units are enabled. This prevents the early, highly-surprising trials from clogging the whole pair space with unimportant or redundant pairs.

A connectionist implementation of incremental enable was achieved by replacing the learn unit with a chain of learn units. Each enables some subset of the pair units. Each is itself enabled by strong activity from pair units that have already been enabled. (Once enabled, a learn unit responds to concept unit surprise as before.) So partition n of the pair units does not wake up until partition $n-1$ has been enabled long enough to have established some pair units. The amount of delay is adjustable.

3.4.4 Performance

Several networks have been built and run on test data using the Rochester Connectionist Simulator [Feldman *et al.*, 1988]. The evidence combination rule used for all these examples is max. The pair units were partitioned into four sets, which are incrementally enabled as described in section 3.4.3.

An easy one

Two data files are used to control the simulations. The first describes the structure of the network; the second describes the learning trials to be run. Here is a data file for a simple network:

```
20 5 0 0
c1 c2 c3;
color: red green blue white;
dots: 1 2 3 4;
scale: small medium large;
shape: tri squa pent hex;
tilt: none left right;
```

The first line has four numbers. The first two specify

1. The number of level-one pair units.

2. The number of attribute-value units with connections to each *site* of a *recruit unit*

The second two are for level-two pair units, which were not used in this experiment.

The second line contains the names of the concepts. This example has just three. The remaining lines contain the names of attributes and their possible values. This file is read by a program which builds the network. Another program is called to run several trials. It reads a data file like the following:

```
c1 is tilt.none 100 scale.small 100, 20;
c2 is color.red 100 dots.4 100, 20;
c3 is shape.squa 100 color.green 100, 20.
```

Each line specifies a concept definition. Line one defines a c1 to be any small object with no tilt. The numbers following the attribute values specify the reliability of the preceding value. All the values in this example have accuracy 100, which is perfect; there will be no noise in the data. The final number on each line represents how many examples of each concept to present (see below).

This is a lot of information to pack into data files, but it makes experimentation very easy. It takes just a couple of seconds to design a new network and/or concept definitions and start a simulation, and descriptions of old networks and trials can be kept on file.

The trials are generated stochastically. On each trial, a definition is chosen with probability proportional to its total number of runs. So, in the current example, each definition has a one-out-of-three chance of being chosen. The actual number of trials

run may not be 20 each. There will be 60 trials total; each definition will be chosen for about 1/3 of the trials.

Once a definition is chosen, the defining attribute values are activated. For each remaining attribute, a value is picked at random (these values are checked to make sure that the input does not also satisfy some other definition). The network is simulated for a few steps to allow activation to reach the concept units. Then external input to the correct concept unit is activated and the network is run for a few more steps to allow learning to occur.

After running the above example (20 trials each), learning was disabled and another 20 trials were presented to test the network's behavior. All 20 were correctly classified. Closer examination of the network showed the following strong inputs to each concept unit (recruited pair units are written as "value ∧ value"):

c1	c2	c3
blue	red ∧ 4	green ∧ square
white		
small ∧ notilt		

The other weights were fairly small. Notice that all the appropriate pairs were recruited. The weights from blue and white to c1 are strong because they are perfect predictors for c1: they can never be on for the other concepts, as those colors are fixed at red and green.

The next test uses the same network, but includes noise in the data. Each of the defining attributes has a 20% chance of being replaced by some other value (chosen randomly). This means there is a 36% chance that a given trial will be incorrect. When there is noise, no double check is made to insure the data does not also satisfy some other definition. After 40 passes over noisy data, with 79 trials being accurate and 41 being inaccurate, the network tested fairly high on two of three concepts. The score for c1 was 37 of 39 correct, for c2 was 37 of 38, and for c3 was 31 of 43. After 35 more passes over the noisy data, the network tested perfectly on c1 and c3, missing just 1 out of 38 on c2. These results are typical of several trials run.

Disjunctive concepts

The next example illustrates the learning of a disjunctive concept. For simplicity, the attributes and values do not have meaningful names here. There are five concepts, which are defined in the training file:

Network description	Concept definitions
30 5 1 0	c1 is a1.a 100 a2.b 100, 30;
c1 c2 c3 c4 c5;	c1 is a1.b 100 a2.a 100, 30;
a1: a b c d;	c2 is a3.b 100 a4.b 100, 30;
a2: a b c d;	c3 is a1.a 100 a2.a 100, 30;
a3: a b c d;	c4 is a2.d 100 a4.d 100, 30;
a4: a b c d;	c5 is a1.b 100 a2.b 100, 30.

C1 is the disjunctive concept. This is indicated by giving C1 two definitions. The definitions of C3 and C5 are meant to confound any simple encoding of C1. This is

similar to exclusive-or. After 30 trials for each definition, the network was tested. It performed perfectly. Here are the strong inputs to each concept:

c1	c2	c3	c4	c5
a1.a ∧ a2.b	a3.b ∧ a4.b	a2.a	a2.d ∧ a4.d	a1.b ∧ a2.b
a1.b ∧ a2.a	a2.c			

The definitions are what one would hope for with the exception of C3. The only strong input is from a2.a, and it has magnitude 0.7. The others are near 1.0, as the probabilities are actually 1.0. This coding works for the current set of concepts, but is not robust. The network assumes tentatively that anything with a2.a is a C3. If a2.a is active but the current concept is not C3, then the correct concept will receive evidence stronger than 0.7.

The weight 0.7 is not very close to the actual probability. This is because insufficient activation of c3 causes surprise, which causes a large positive weight change from a2.a. The weights will not represent approximate probabilities when the representation is inexact. They may grow too large or small in order to improve performance. The above example was run again with a new random number seed, so the connectivity pattern and learning trials would be different. The second time, all necessary pairs were recruited.

The same network was run with the same definitions, but with noisy data. Each defining attribute had 10% chance of being wrong. Since the above definitions have two parts, each with a 10% chance of error, the chance for an error on each trial is 19%. After one round (30 examples of each, including noise) the network was tested. The only errors were for c2 and c3, which missed 15 out of 60 together. After another round of noisy training, there was only one error for c4 out of 30. The others had a perfect score. The main problem with noisy data is the recruitment of junk. Noisy input will cause surprise thus initiating the recruitment of unimportant pairs and using up resources.

Instantiation of features by concepts

Learning works both ways: from the features to concepts and from the concepts to the features. Attribute-value units respond weakly to top-down input from the concepts and pair units. This is reasonable as default feature inferences are weaker than direct perceptions. It also prevents runaway feedback loops. This section presents some experiments conducted on the network which learned the concepts presented in the previous section. First, values for only three of the attributes were activated: a1.c, a2.d, and a3.c. These were chosen as typical values of c4, which was defined to be (a2.d and a4.d). After a few steps, c4 had activity 0.57. This activated a4.d, which had the potential .25. Even though it was weak, it was the network's best guess for an a4 value.

If no external input is provided for the attribute values, and c3 receives external input, it turns on the following:

a1.a	.48	a4.a	.061
a2.a	.48	a4.d	.085
a3.a	.17		

This process breaks down in the case of disjunctive concepts. The two disjuncts tend to become mixed together. My original thought was that the inhibition at the attribute-value level combined with the mutual reinforcement of the attribute-value and pair units would draw out the more probable definition, but this does not occur. Here is the result of activating the disjunctive concept c1:

a1.b	.35	a3.a	.20
a2.a	.35	a4.a	.15

In this case one of the disjuncts is selected consistently, but a more perverse set of definitions can be designed so that the most probable individual attribute values are from different disjunctive terms.

Learning longer definitions

So far, all the definitions have involved pairs of attribute values, which meshes perfectly with the pair units, and the network has performed well. Long definitions, especially those with a large between-definition overlap in features, pose a much tougher test. In order to better handle longer definitions, high-order pair units are used. These units respond to one input from the attribute values and one from the first-level pair units. The following data file specifies a network with 100 pair units connected to 12 attribute-value units at each site and 50 triple units connected to 12 attribute-value units at one site and 12 pair units at the other site.

Network description	Concept definitions
100 12 50 12	c1 is a1.a 100 a2.b 100 a3.c 100 a4.d 100, 30;
c1 c2 c3 c4 c5;	c2 is a1.a 100 a2.b 100 a3.c 100 a4.f 100, 30;
a1: a b c d e f;	c3 is a2.e 100 a5.c 100 a6.a 100, 30;
a2: a b c d e f;	c4 is a1.c 100 a6.b 100 a7.c 100, 30;
a3: a b c d e f;	c5 is a4.a 100, 30.
a4: a b c d e f;	
a5: a b c d e f;	
a6: a b c d e f;	
a7: a b c d e f;	

The results:

after 150 trials	after 300 trials
c1 missed 6 out of 37	c1 missed 0 out of 29
c2 missed 8 out of 27	c2 missed 3 out of 32
c3 missed 5 out of 24	c3 missed 0 out of 27
c4 missed 4 out of 33	c4 missed 2 out of 24
c5 missed 2 out of 29	c5 missed 1 out of 38

There are too many units involved to report on them all in detail, but some of the more interesting pair units recruited include:

for c1: (a1.a ∧ a4.d) (a3.c ∧ a4.d) (a2.b ∧ a4.d)
for c2: (a2.b ∧ a4.f) (a1.a ∧ a4.f) (a1.a ∧ a3.c ∧ a4.f)
for c3: (a2.e ∧ a5.c ∧ a6.a)

The final test reported here uses a network with seven attributes, each with six values. There are nine concepts, 100 pair units, and 50 triple units. As in the previous example, there are 100 pair and 50 triple units. The definitions of c1 and c2 were designed to be very confusing:

Network description	Concept definitions
100 10 50 12	c1 is a1.a 100 a2.b 100 a3.c 100 a4.d 100, 30;
c1 c2 c3 c4 c5 c6 c7 c8 c9;	c1 is a1.a 100 a2.b 100 a3.e 100 a4.f 100, 30;
a1: a b c d e f;	c2 is a1.a 100 a2.b 100 a3.c 100 a4.f 100, 30;
a2: a b c d e f;	c3 is a2.e 100 a5.c 100 a6.a 100, 30;
a3: a b c d e f;	c4 is a1.c 100 a6.b 100 a7.c 100, 30;
a4: a b c d e f;	c5 is a4.a 100, 30;
a5: a b c d e f;	c6 is a6.f 100 a7.e 100, 30;
a6: a b c d e f;	c7 is a2.c 100 a4.c 100 a5.a 100, 30;
a7: a b c d e f;	c8 is a3.e 100 a4.b 100 a6.b 100, 30;
	c9 is a1.c 100 a7.f 100, 30.

The score for the second round of 300 presentations is:

c1 missed 0 out of 38	c5 missed 1 out of 45
c1 missed 5 out of 32	c6 missed 1 out of 26
c2 missed 3 out of 33	c7 missed 0 out of 26
c3 missed 0 out of 24	c8 missed 7 out of 15
c4 missed 5 out of 32	c9 missed 10 out of 29

Some of the more interesting pair units recruited include:

for c1: (a2.b, a3.e) (a3.e, a4.f) (a1.a, a3.e) (a2.b, a4.d) (a3.c, a4.d) (a1.a, a4.d) (a2.b, a3.e, a4.f) (a2.b, a3.e, a7.f)
for c2: (a1.a, a4.f) (a3.c, a4.f) (a2.b, a4.f) (a3.c, a5.e) (a2.b, a3.c, a4.f)
for c9: (a1.c, a7.f)

Repeated simulations from new configurations continued to perform less well than might be expected. The triple units do not reliably select key triples. These experiments are discussed further at the end of the chapter.

3.5 Enhanced Competitive Learning

The pair units perform poorly on long definitions. Long, complex definitions are the most interesting, so it is important to deal with them efficiently. The pair units can only build representations bottom-up, starting with pairs, and from these making triples or quadruples, etc. A simpler approach to recruiting representations of arbitrary n-tuples is to use a single hidden unit. In the implementation described here, this is made possible by totally connecting the attribute-value units and the hidden units. This total connectivity will not scale to realistic domains. Still, experimenting with this approach is useful. There are various possible solutions to the connectivity problem involving paths rather than single links [Feldman, 1982].

The question now becomes, how to control the response of the hidden unit. The pair units only get a "surprise" signal; they select their individual pairs based upon the random pattern of their connections and the frequency of occurrence of the pair.

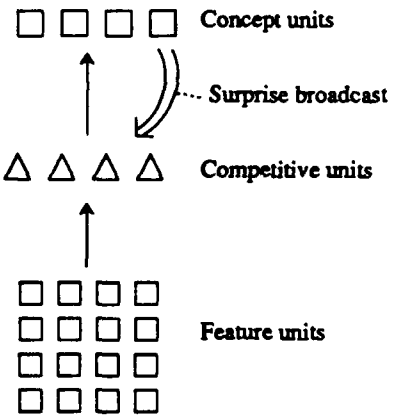

Figure 15: The middle layer consists of competitive learning units.

This sort of strategy will not work with totally connected hidden units. If they are to function with only a surprise signal (no link-specific feedback as with back propagation, for example), some force must prevent them all from representing the same thing. If the hidden units compete in a winner-take-all network so that only one responds to each input, we have something very like competitive learning [Grossberg, 1976b, Grossberg, 1976a, Rumelhart and Zipser, 1985]. The difference being that competitive learning is unsupervised, and here the hidden units provide support for concept units which receive direct feedback and change their weights accordingly.

The setup illustrated in figure 15 was implemented to test the idea of using competitive learning for the hidden units in a supervised learning task. The competitive learning is enabled by the surprise broadcast; the motivation being to provide the competitive units with at least some information about their progress. With the surprise broadcast, learning is focused on those inputs presenting difficulty. The implementation of competitive learning used differs from Rumelhart and Zipser's, though the effect is similar. Rumelhart and Zipser keep the total sum of incoming weights constant; the weight is redistributed when a unit wins a competition so that active inputs get more and inactive ones get less. Achieving this with local weight-change rules is somewhat tricky; the weight-change function at each link needs to know exactly how many other links are active and how many total links there are before it can make a weight change.

Here, the winning competitive unit also increments the weights on active and decrements the weights on inactive inputs, but no attempt is made to keep the sum of weights constant. Instead, the units have a floating expectation. Every time the sum of inputs exceeds the old expectation, it is incremented to equal the new sum. Each unit's activity is equal to the percentage of the expectation which has been realized. To keep the weights from growing without bound, an absolute ceiling is set at unity.

Experiments show that some competitive units would never win a competition because their randomly assigned initial weights are too far from any of the input patterns. I have adopted one of the solutions proposed by Rumelhart and Zipser: Each time a competitive unit loses, it lowers its threshold a little. This eventually sensitizes it enough to allow it to win. However, this leads to very unstable behavior when the number of competitive units is greater than the actual number needed. Units which don't find a

niche become increasingly sensitized until they beat other units whose support matches the input much more closely. Since the concept unit has by then learned to depend on the other hidden unit, the input will be misclassified even though the network may have learned the definition almost exactly. This problem is dealt with by only lowering the threshold on losing competitive units when there is surprise and no other competitive unit is responding strongly. In addition, there is a limit on how low the threshold will go.

For concept definitions which are completely specified with little overlap between the concepts, the network works well, needing only about eight passes over the data to achieve perfect performance. For each component of the definition, a hidden unit learns to respond to it exactly and the concept unit responds to that hidden unit. However, it fails when the definitions are more complex, with some features unspecified and others overlapping with other concepts.

The failure of this method for more complex definitions is not surprising. Competitive learning has no way of knowing which features are important and which are not; they are all treated equally. If two concepts have similar definitions, it is possible for objects in different classes to be superficially more similar than objects in the same class. It is instructive to analyze how other learning techniques avoid this problem. A_{rp} [Barto and Anandan, 1985] does a stochastic search; units which distinguish the important features are rewarded. However, stochastic search is not compatible with the goal of quick learning. Back propagation sends individual reinforcement to the hidden units. These signals can be used to "choose" different hidden units for different concepts, no matter how similar.

The competitive learning network described above was modified slightly in order to achieve a similar top-down isolation of concepts. No special mechanism was used; links were added from the concept units to the competitive units. The effect of this is to make two inputs which belong to the same concept appear more similar to the competitive units insofar as they share this top-down activation. The competitive units receive bottom-up activation in the range 0 to 1000, with 1000 meaning all the expected activation has been received, as explained above. The top-down activation is added to this total. This can be as much as 500, so the top-down input contributes one-third of the total possible activation to the hidden units.[3] This top-down link makes a significant difference in the performance.

The weight change rule for links from competitive units to the concept units is the same as that used in the previous network to approximate the probability of the concept given activity of the hidden unit. When the the hidden units successfully partition the input space, the weights are high to the corresponding concepts and low to all others. The exact nature of this weight change rule is unimportant. Any bounded Hebbian rule will work.

Without the top-down links, the following definitions are unlearnable. With them, the network achieves perfect performance in fewer than twenty passes over the data. (The definitions are in terms of attributes and their values.)

[3]The exact magnitude of this feedback is much more important in the structured concept learning network described in chapter 4. See section 4.3.6.

c1 is a1.a a2.b a3.c a4.d
c2 is a1.a a2.b a3.c a4.e
c3 is a2.d a5.c a6.a
c4 is a1.c a6.b a7.c
c5 is a4.d a5.c

The next run uses the easily confused definitions given in section 3.4.4:

c1 is a1.a a2.b a3.c a4.d
c1 is a1.a a2.b a3.e a4.f
c2 is a1.a a2.b a3.c a4.f
c3 is a2.e a5.c a6.a
c4 is a1.c a6.b a7.c
c5 is a4.a
c6 is a6.f a7.e
c7 is a2.c a4.c a5.a
c8 is a3.e a4.b a6.b
c9 is a1.c a7.f

This pushes the ability of the network to the limit. On about half the trials similar disjuncts of c1 and c2 are be separated into their own competitive units. Other times, this separation fails and one or more category be consistently misclassified. The singleton c5 caused the most trouble. The large variation among its instantiations hinders recruitment about half the time. Such short definitions would be more easily handled if there were direct links from the features to the concept units. The other concepts virtually always recruit a correct hidden representation.

When a concept is activated, it will excite strongly connected hidden units, which excite feature units in turn. A disjunctive definition, such as c1 in the previous example will instantiate one disjunct or the other without confusion. This is because the two hidden units representing c1 are mutually inhibitory; only one will remain on. Only its features will be activated. This is a big improvement over the performance of the pair units, which could easily confuse disjuncts, and and even bigger improvement over back propagation. Back propagation is more powerful in the forward mapping, but cannot handle the backwards mapping simultaneously, as no loops are allowed.

3.6 An experiment with back propagation

As described in section 3.3, back propagation is a general purpose learning algorithm for multi-layer networks. It has been successfully applied to several different tasks. As exciting as these results are, there are limitations inherent to *any* general purpose algorithm. It is asking too much for a single, ignorant (i.e., without domain knowledge) technique to generalize optimally in different domains. There is no reason to expect the regularities of different domains to have the same nature. Since the nature of back propagation's abstraction is the same for all domains, there may be a problem.

This section explores the nature of the generalizations back propagation makes in a difficult task, and describes various attempts to influence those generalizations. The task is a kind of pattern recognition meant to be analogous to natural language in a *very* simple way. All inputs consist of 5 consecutive words. There are 16 possible words

in each position, for $16^5 = 1,048,576$ total possible inputs. The words in each position are grouped into 4 classes of 4 words each. Permissible input patterns are defined in terms of these word classes. There is only one output unit, which should have a high value for valid inputs and a low value for invalid ones. The task is nontrivial because the network must learn the word classes and legal patterns at the same time based solely upon correctness feedback.

The set of legal sentences used to train all the networks is depicted below (the boxes will be explained below).

[A]	B	C	D	D
C	A	[B]	[B]	C
A	D	[C]	B	A
B	C	A	C	[D]
C	[A]	B	C	D
A	[B]	C	D	A
[C]	A	D	B	B
C	B	B	[D]	B

The letters specify a word class in each of the 5 positions. The pattern "A B C D D" is meant to be analogous to, say, "DET ADJ NOUN VERB ADV." There are four words in each word class: word class A includes words A1 through A4, word class B includes words B1 through B4, etc. The training alternates between valid and invalid patterns. Valid patterns are picked from the set of legal patterns randomly. The invalid patterns are generated from a random valid pattern by changing one of the five inputs. This is the simple training regime.

In order to test the generalizing power of the learning algorithm, the restricted training regime omits certain inputs. For the word classes represented by the boxed entries, only half of the words in that class are used during training, and only the other half are used to test the network. This means that no input used for testing ever appears in training. What is more, for each sentence, there are words which are never used during training. The network might generalize to the never-seen words based on the use of all the words in the same class for other sentences in the same position. For example, the training for the sentence [A] B C D D will never include the words A1 and A2 in position 1, and will have only those words during testing. If the network is to recognize these words as part of a legal pattern, it will be because of the sentences A D [C] B A and A [B] C D A, which use all the A-words during training. The "domain knowledge" needed to perform this task is this: if two words belong to the same class for some sentences, then they belong to the same class for all.

Three different network architectures were used. Figure 16 depicts the unstructured network. It consists of a single layer of hidden units sandwiched between the input and the output. I conjectured that such a network would not perform well in the restricted task because there is no reason for the solution to separately represent word classes. Figure 17 shows a more complex network with two hidden layers. The first contains four units per position, which are only connected to input units in the same position. There are just enough units to have one per word class. The second hidden layer is

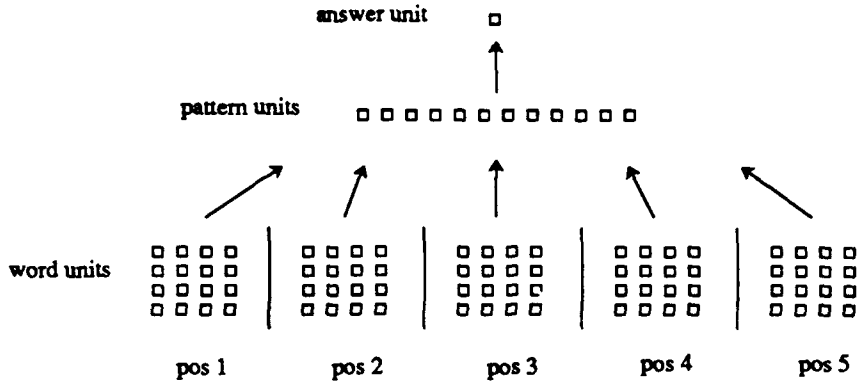

Figure 16: Single-layered setup for grammar learning task.

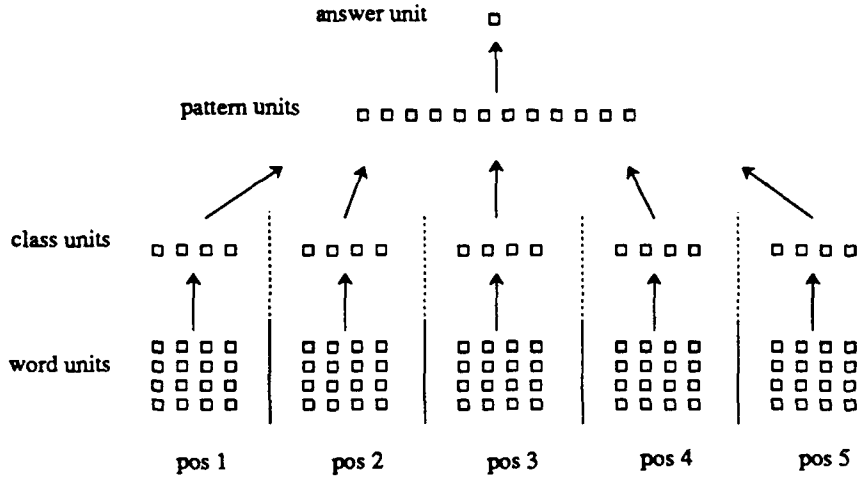

Figure 17: Multi-layered setup for grammar learning task.

connected to all and only the first hidden layer. The idea is that, for each position, all information about the input must be represented by the state of four hidden units. The only information about the input which counts is which class they are in. This is designed to encourage an explicit representation of the word classes, resulting in correct generalization in the restricted training task.

The results are summarized in table 7. Four different connectivity patterns are used: single and double hidden layers, as described, with links only to the next highest layer *and* with links to all higher layers, including directly from the input to the output. The total connectivity condition is indicated with a "+" in the table.

The table entries show the number of misses out of 500 trials for both invalid and valid input. For the restricted training trials, false negatives predominate, which is not surprising given the nature of the training. The performance of the single layer network is surprising. After 100,000 trials, about 90% of the novel test inputs are correctly

Table 7: Errors out of 500 (false pos./false neg.).

training	connectivity	number of trials 1000	5000	10000	20000	50000	100000
restricted	single	500/497	125/437	47/303	23/221	0/67	0/56
	single+	403/482	246/477	155/375	53/196	48/67	17/56
	double	500/500	362/500	239/500	6/68	0/0	0/0
	double+	431/396	106/497	53/272	20/257	0/188	0/118
simple	single	500/500	230/355	58/3	0/0	0/0	0/0
	single+	430/472	289/409	118/18	67/0	59/0	40/0
	double	500/500	498/500	377/265	0/0	0/0	0/0
	double+	448/376	112/321	88/0	27/0	21/0	21/0

Table 8: Thresholds and support for class units in position three.

	Unit 1				Unit 2				Unit 3				Unit 4			
thresh	-0.2				-0.7				0.0				-0.1			
class one	-2.0	-1.3	-1.3	-1.2	2.0	2.6	2.7	2.8	-2.0	-2.0	-1.9	-1.8	0.5	0.9	1.0	1.1
class two	1.3	1.5	1.5	1.4	-1.7	-1.1	-1.2	-1.3	1.3	1.7	1.6	1.5	-1.7	-1.4	-1.4	-1.5
class three	-1.4	-1.5	-0.5	-0.3	-1.5	-1.6	-2.1	-2.4	1.0	1.0	0.4	0.3	1.6	1.4	2.2	2.5
class four	0.7	0.2	-0.4	0.4	-0.2	0.0	0.2	-0.1	-1.1	-0.7	-0.1	-1.0	-0.6	-0.9	-1.2	-0.9

classified.[4] The network is generalizing over word classes, albeit imperfectly. Human interpretation of the final network is not easy; it was never determined what solution strategy was used. The two-layered network designed to encourage such generalization does perform better. It achieves 90% performance after only 20,000 trials, and perfect performance after 50,000. It is interesting that adding additional links to the two layer network significantly hampers performance. The network is no longer forced to squeeze the input through the four "class" units. The initial learning curve is faster (see the 10,000 trials entry), but the final result is not as good.

Even though the structured network generalizes perfectly, the solution it finds is not the straightforward, localist one I envisioned. Never did the four class units in a position find the solution where each represented a single word class. The connectivity of the four class units in position three for the structured network is shown in table 8. Clearly the words are being divided into classes, but not in a simple one-to-one manner. This is an example of coarse-coding [Hinton, 1981b].

Two other restricted training regimes were tried. For the restricted word classes, instead of using two words in training and two in testing, one regime used only one in training and the other three in testing and one used three in training and the other one in testing. When only one word was used in training, the test performance for all configurations was very poor. When three words were used in training, the performance of all configurations went up, with the structured network achieving perfect performance after 20,000 trials and the single-layer network leveling off at 5% false positive and no false negative.

[4] Training to 200,000 trials shows no further improvement.

Several efforts were made to force the representation found by the structured network to take the easily interpreted form: one class unit responds to each class; one pattern unit responds to each pattern. This would have insured generalization in the restricted task. We also hoped to speed convergence by biasing the solution. One effort restricted the bottom-up weights to be positive and the unit biases to be negative. This was unsuccessful. The network did not perform well. Another effort imposed a winner-take-all structure on the class units in each position and on the pattern units. This was done by augmenting the error value of the non-winners to encourage them to be zero. This also failed.

3.7 Discussion

The pair-based learning scheme described above met with some success, but encountered a number of difficulties. A discussion of its problems may prove useful, but I will first mention some of the good properties. Most important, for a large class of definitions, it works. It is also fairly fast, and the noise tolerance is good. The learning is permanent, even if training on new definitions begins. The network can be interpreted by human users: the hidden units always represent pairs (or triples, etc.). The network runs backwards: if a concept unit is activated, it will in turn activate attribute values according to how typical they are for that concept. Although external feedback is necessary for the recruitment of hidden units, once a concept is learned, the weights from recruited hidden units and attribute-value units will continually adjust their values even without feedback.

The network has a number of problems. There is no way to reclaim useless pair units. Inevitably, pair units are recruited to represent unimportant pairs. Since the number of pair units is limited, this is a problem. Also, there is no obvious way to prevent several pair units from representing the same pair. This problem is magnified with the higher-level pair units. A very small percentage of them manage to learn something useful. They would often be recruited to represent triples such as *large and red and large*. This repetition was prevented in the bottom-level pair units by not allowing two links from the same unit. This is not sufficient in the higher levels as it is a single value may be represented at several lower-level units. This problem is greatly exacerbated by the presence of redundant units at the lower level. After dozens of experimental variations in connectivity and weight-change rules, it is clear that hierarchies of pair units are not the answer to learning hidden representations of long conjunctions.

In part because of the above problems, summing the input at the concept level does not work. Using a sum, noise can easily overwhelm the important inputs. This is exacerbated by constraining weights to be approximate probabilities. The weight from an important input cannot grow as large as necessary to exceed the noise. The motivation for approximating probabilities with the weights was the desire to have understandable weights and to make it possible to adjust them even in the absence of feedback. A delta rule only works with feedback. A simple Hebbian rule would make them grow without bound. The approximate probability rule is essentially Hebbian with a soft bound. The nearer the weight gets to the bound, the more slowly it grows. The probabilities approximated are not accurate enough to use Bayesian combination

rules, especially since the independence of the inputs is not know.

Taking the maximum works well in those cases where the concept has an exact definition which is represented by a single unit. However, because of the poor performance of the higher-level conjunction units, this is not likely for long definitions. Even if the higher-level conjunction units did work well, using max forces the network to be extremely punctate, with each unit attending to only a single input. This is unacceptable and robs connectionist models of much of their power.

The network was not designed to deal with negation at all. A definition such as *large and not green* could only be represented as an ungainly disjunctive definition *large and blue* or *large and red* or ...

Development of the enhanced competitive learning algorithm followed from the frustrating results using pair units and takes a much more direct approach. It was more successful. Clearly the strong top-down feedback helps to isolate important conjunctions of features. It is interesting how so small an addition can help performance. The idea is similar in motivation (but different in implementation) to Lynne's [1988] competitive reinforcement learning, which augments reinforcement learning to achieve a kind of competitive learning which can be influenced by other forces, such as the top-down feedback presented here. This was not used because of the slowness of reinforcement learning's stochastic search and because the links to the competitive units in Lynne's scheme base their learning on a recent activity trace. This will not work in the dynamic environment used in chapter 4; with some additions, the scheme described here does.

The top-down activation to the competitive units does not provide nearly the detailed information that back propagation does, and the technique is not as powerful, as evidenced by its partial failure on the tricky concept definitions—definitions back propagation would almost certainly handle consistently. However, the ability to learn with no specific error information proves very useful in the work presented in chapter 4 which uses this modified competitive learning as a component in a more complicated structured learning task.

The back propagation results do not contribute in any significant way to the other work in this thesis. Back propagation is still the most powerful concept learning technique, so exploring its behavior and abilities is interesting. It is impressive, but not surprising that the structurally biased back propagation network is able to solve the parsing problem and generalize to unseen inputs. What is more surprising is the extent to which the single-hidden-layer network also made these generalizations. It is still not clear what lesson can be learned from this as the exact nature of the solution found was obscure. Significantly, the two-hidden-layer network performed less well when additional links. The constrained connection pattern forced the solution to take a form which generalized in the desired way. Other attempts to influence the solution found were disappointing. It would prove useful if a single, powerful weight change algorithm could be easily customized to various special domains.

Chapter 4

LEARNING STRUCTURED OBJECTS

4.1 Introduction

Even representing structured objects in connectionist networks is problematical. A simple object description consisting of some features can be easily represented by activating the relevant features. However, two such objects cannot be simultaneously represented by activating the features of each because of cross talk. There is no way to tell which features belong to which object. Figure 18 illustrates the problem. It is impossible to determine whether the objects represented are a medium square and a large triangle or a medium triangle and a large square. A structured object has multiple parts with their inherent cross talk potential as well as relations between the parts which is an additional source of cross talk.

This is a very important limitation of feature vector representations. Some researchers [Anderson, 1985, Hinton, 1981a] have circumvented the problem by using more than one representation space as illustrated in figure 19. This does eliminate cross talk, but at too great a cost. Taken to extremes, this strategy would necessitate duplicating the entire representation space as many times as there are objects to be considered simultaneously. When we add temporal reasoning and relations, the whole thing blows out of proportion. We not only have several objects to contend with, but we must also consider several states of each. At some level, concepts must be separated in a more sophisticated manner than with redundant representation spaces.

A further complication is introduced with learning. If there is a change in one representation space, e.g., a new fact about John is learned, then that change must be propagated to all the representation spaces. McClelland [1985] invented a way to

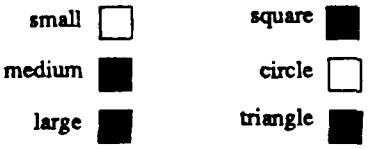

Figure 18: Multiple objects cause crosstalk.

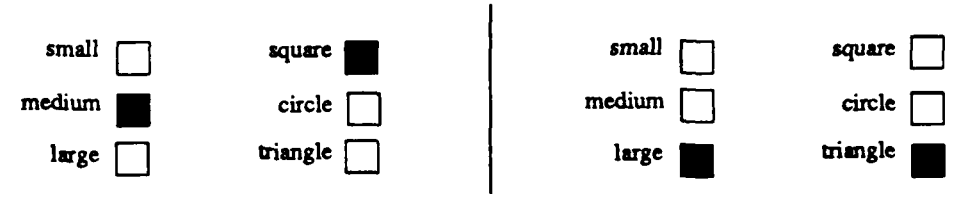

Figure 19: Multiple representation spaces stop cross talk.

do this, but it is too resource-intensive to be practical. Chapter 2 described another technique, but it is too inflexible for general purpose use.

In an extremely punctate representation, cross talk is eliminated by using very specific units. Instead of representing a large square with two nodes, there is a single node meaning *large-square*. The two nodes *large-square* and *medium-triangle* can be simultaneously active with no crosstalk. Unfortunately, such representations are totally unrealistic [Feldman, 1986].

Cross talk must be avoided. Separating the representations spatially is not practical. Therefore, they must be separated temporally. The strategy I propose is to process the components of a complex object sequentially. This need not result in the loss of parallel processing. Each component of the object can have an arbitrarily rich representation which is processed in parallel. Simultaneous low-level activation of all components is allowed, and this can prime or otherwise bias the current computation. When the complex object is categorized, a high-level representation of the entire object will be activated which represents that object as a whole and which can take its turn in further reasoning. The major problems to be solved are the sequential combination of evidence, delayed credit assignment,[1] and dynamic control of attention, i.e., determining which subparts to process when.

For humans, sequential processing is clearly necessary for high-level cognition. This does not mean that the granularity of sequentiality I propose is correct, of course. Certainly there are some resource-intensive structures in the human brain to aid parallel processing, such as in low-level vision. I am hypothesizing that reasoning in general requires a sequential component at a finer grain than has been previously suspected by connectionist researchers. This chapter first sketches these ideas in more detail, then describes experiments with a preliminary implementation of a network which learns to classify simple structured objects in a position independent manner.

4.2 Representing Multiple Objects and Relations

I have proposed separating multiple objects in time instead of space. During processing, they must be focused upon one at a time. The representation of relations is like that proposed by Shastri [1984]. In order to distinguish *John loves Mary* and *Mary loves John*, *loves* consists of *agent-of* and *object-of* parts. Each part is bound to the individual

[1] I do not use the term "temporal credit assignment" as that has a different meaning and is much more difficult. The problem referred to here is not determining which of a sequence of events should be rewarded or punished. There is a single event. The problem is accomplishing the credit assignment when the event is broken into subparts which are not simultaneously present.

filling that role. As attention is focused upon each individual in turn, their roles in the various relevant relations, as well as all their properties, are also active. Dividing relations into two parts helps in the implementation of rules and binding as outlined in chapter 5. The parts of a relation must be closely bound together. Mary may love John and Bill and Sue may love Bill as well. Each fact must be stored separately in such a way that it is not confused with related facts. The relations are organized into something like a type hierarchy. Their inherited properties control inferences. For example, all touching relations require proximity. Much as with objects, relational representations are distributed in so far as the representation for any particular relation consists of the activation of many properties, superior relations, etc. Some units specific to that relation are also necessary to facilitate control.

The first issue which arises is controlling the switch of attention. It is important to do this within the connectionist paradigm as opposed to using external modules. Controlling attention is an important part of cognition. The level of attention involved here is relatively simple, but mechanisms which prove useful for this task may form the foundation for high-level attention. Given a complex object, attention must switch from primitive to primitive. The description of each primitive includes the activation of many units (features, etc.). In order to coordinate all these units, some overhead is necessary. Special control units must dynamically bind to each primitive's description. These function as a temporary handle. When the control unit is active, the bound description is activated. Inhibition between the control units insures only one is active[2] at any time. Exhaustion of the control unit after some number of steps will induce a switch of attention as the next strongest control unit subsequently activates. At any time, there may be many concepts fighting for attention, i.e., competing for the limited number of control units.

An important problem which immediately appears is the summing of evidence over time. All the evidence needed to support a complex concept may be present, but it is not active simultaneously because the components are attended to sequentially. Sequential evidence combination in connectionist networks using recurrent connections has been used by several researchers recently [Jordan, 1985, Watrous and Shastri, 1987, Elman, 1988, Pollack, 1988]. The general idea of this work is to modify a three-layer back propagation network so that the outputs of the hidden units at time t become part of the input at time $t+1$. A sequence of inputs is provided. The recurrent connections augment the input with state information which represents, in condensed form, all the previous inputs. The results have been interesting, but it is too soon to know the full power and limitations of the technique. I propose trying a different, more explicitly structured approach. The idea is to specifically hang on to the (exact) sequentially presented components as long as necessary. They are focused upon repeatedly, if necessary. Somewhere, the evidence they provide for each concept is stored in parallel. In the implementation described below, links "remember" recent activity and in this way provide, simultaneously, input from all primitives recently active. More powerful solutions involving storage of recent evidence using extra units will be necessary. The feasibility of explicitly retaining all components of a more complex concept goes down as the complexity of the concept goes up. Some sort of selective forgetting is necessary.

[2]By active, I mean strongly active. Low-level activation of other control units is not precluded.

Most likely, the components need to periodically combine to form higher-level representations in a hierarchical fashion. If too many components are encountered before they can be chunked, information will be lost.

In order to teach a network to recognize a complex object, the primitives are presented one at a time, along with their interrelationships. Then feedback is provided as to the correct concept. Explicitly remembering all the primitives makes it easy to separately reward them. After the feedback, further cycles of attention present each primitive anew for appropriate weight adjustment. Once again, as the complexity of the concept increases, the feasibility of doing this decreases. The problem of temporal credit assignment remains. It will be necessary to learn a bit at a time, chunking more and more complex concepts as "primitives" for more complex learning.

There are several potential difficulties with the scheme as outlined, but none are obviously fatal. It may seem that by processing the primitives of a complex object sequentially we lose the ability to have a gestalt of a complex object. My hope is that the high-level representations mentioned above will serve this role. Certainly high-level representations are necessary, otherwise complex concepts would be impossible. In order to contemplate the concept *argument*, one must consider the concept *person*. A person is composed of beliefs, thoughts, legs, arms, etc.; arms have hands; hands have fingers; fingers have knuckles; and so on. Clearly, if the subparts of a concept must be focused upon sequentially, then it must be possible to think about a concept without thinking about its parts. Even ignoring the sequential nature of the process, it seems unlikely that thinking about an argument activates the representation of knuckles.

Another difficulty involves the cross-talk arising from even low-level activation of background concepts. It is not clear how to distinguish low-level activation due to background objects from low-level activation due to weak evidence in focused-upon objects. For example, suppose we are pondering a bird with an unknown color on a red bird feeder. We are focusing our attention on the bird. Because of residual activation due to the bird feeder, red is somewhat active. We don't want to confuse this with the bird being a little red or with there being some evidence for the bird being red. The next section presents a first attempt at realizing these ideas in a working network.

4.3 A Preliminary Implementation

4.3.1 The problem

The implementation described in this section follows the discussion above for the most part, but makes some simplifying assumptions. In particular, there are no instances or particular relations remembered (such as *Mary loves John*). The task was simplified in order to focus on the control problems stemming from the multiple shifts of attention involved in processing structured objects.

The problem is learning to recognize structured objects presented on a 4×4 grid. Each of the 16 grid locations can contain a primitive object. For simplicity, each primitive consists of values for only two attributes, labeled "size" and "shape." The spatial relations implicit in the grid structure are used in learning the object descriptions. They are: over, under, left and right. Figure 20 shows a large square over a small triangle which is right of a medium circle.

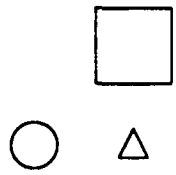

Figure 20: Simple structured object.

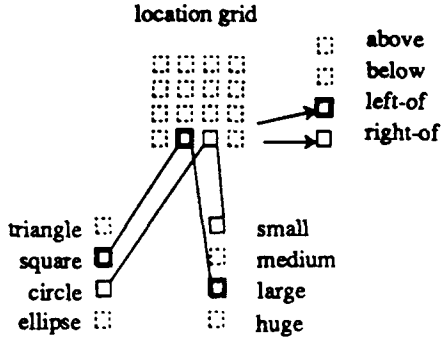

Figure 21: Representation of "large square left-of small circle."

Although the task seems somewhat visual in nature, this network is not doing computer vision. The grid locations do not represent pixels. This is an artificial task designed to demonstrate some techniques for representing and learning relations. The grid is used to induce spatial relations.

4.3.2 Organization of the network

Figure 21 illustrates the representation of multiple objects and relations. The 4×4 matrix of units is the location grid. Grid units are mostly place holders. They do not contain any semantic content beyond location. They also serve to control attention by alternately becoming active. For each primitive in a structured object which is being attended to, there will be a unique, dedicated grid unit. In figure 21 units which are inactive have a dashed outline. Units with a low level of activity have a narrow outline. Units which are fully active have a thick outline. There are two primitives represented: a large square and a small circle. The grid unit corresponding to the square is fully active and is dynamically (i.e. temporarily) bound to the properties defining the square. The grid unit corresponding to the circle is at a low level of activity and so are the bound properties (*circle* and *small*).

The relation unit *left-of* is also fully active; the primitive being attended to is left of another primitive. This relationship is detected by a network of units illustrated in figure 22. Relation detection units are replicated throughout the grid, four per location. More sophisticated ways to detect relations are possible, but not important to the current experiment. What is important is the central representation of each relation seen by the rest of the network. Each *left-of* detector in the grid activates the same, global *left-of* unit. This abstraction away from the specific input location makes the

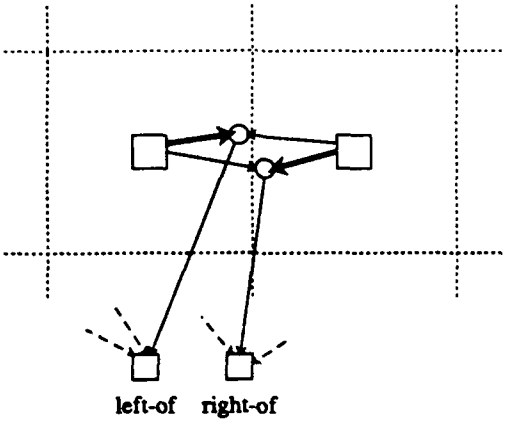

Figure 22: Relation detectors between two grid units.

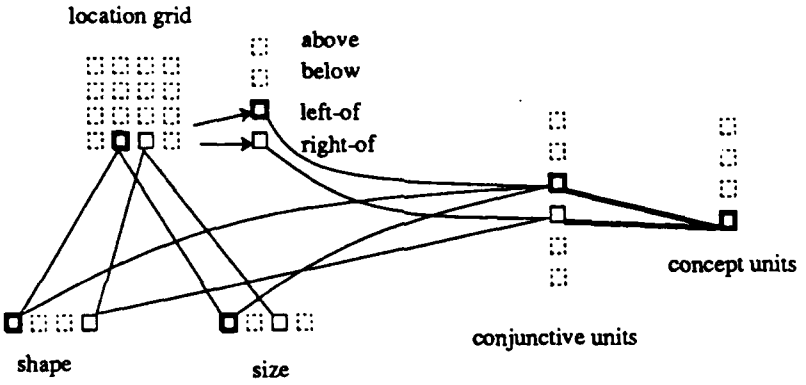

Figure 23: Layout of the entire network.

learning that occurs inherently position independent.

The way that structured objects are input is described below. Once the primitives are bound to grid units, the grid units control the change of attention. When a grid unit is active, it activates the bound properties: that primitive is being attended to. The dynamics of the network will be discussed in more detail below.

Two further populations of units complete the network (see figure 23). The concept units represent the output of the network; they answer the question "what is it?" During training, they receive external activation to indicate the correct answer. They are mutually inhibitory. They receive input from the conjunctive units.

Conjunctive units represent combinations of objects and relations. Initially, they are weakly connected to all property and relation units. Eventually, they will learn to respond to important primitive objects and their relations. If the concept to be learned is *large circle left-of square*, there will be a conjunctive unit responding to {*large* ∧ *circle* ∧ *left-of*} and another responding to {*square* ∧ *right-of*}. Activation from both these units will be necessary to fully activate the concept. Figure 23 shows such a concept being recognized.

4.3.3 Dynamic behavior of the network

In order to avoid cross talk, the objects must be entered one primitive at a time. In order to provide external input to the network, there are special external units corresponding to the grid and properties as well as the concept units. These units represent whatever source of input would exist in an integrated system. They are turned on and off by the experimenter. Using explicit input units makes the operation of the main network more natural. The units only respond to inputs; no special experimenter-induced states are necessary. The external units also provides a kind of connector by which the network can be joined to other networks, such as those that recognize properties.

To enter a large square left of a small circle, one of the primitives is chosen and some location is picked (randomly) for it. Suppose the large square is to be entered first. The external grid unit for that location is turned on as are the external units for square and large. This in turn activates the corresponding internal grid unit and property units. Now the grid unit becomes bound to the active property units. Once the external input is removed, the grid unit continues to hold everything together. This binding is done in the most straight-forward way possible. The links from grid to property units normally have weight zero, but if both the grid and the property units are strongly active (as they are when stimulated externally), then the weight becomes temporarily strong. It will stay strong until the grid unit becomes totally inactive. More sophisticated binding techniques are necessary, if only because directly connecting control units (such as the grid units here) to every unit which may become bound will require too many links. Feldman [1982] discusses some efficient binding techniques.

After a few steps of simulation to allow binding to occur, the next primitive object (the small circle to the right of the square) can be entered in the same way. The external grid unit to the right of the square is activated as are the external circle and small units. The grid units are mutually inhibitory, so when the new grid unit is stimulated, the one previously active is pushed down to a low level of activation. (If it went totally off, the dynamic-bindings would be lost.) External activation is stronger than routine internal activation so that new inputs always suppress the current object of attention. This seems like a reasonable mechanism, although people can certainly "tune out" sensory input during thought. This shifting of attention is somewhat tricky. Before the new input can suppress the old object of attention, it must become strong. For at least a short time, both the old and new are active—presenting a danger of crosstalk. In particular, since the dynamic binding between grid units and property units results from simultaneous activation, there is a danger that the new grid unit will become dynamically bound to the previous, not-yet-suppressed properties. This is avoided by setting the threshold activity for dynamic binding to a level above that of normal internal activity but below that of external activation. As long as the current units have settled to an internally active state, they will not participate in any dynamic binding with new inputs. The problem of confusion during a switch of attention will certainly come up again. The solution used here relies on activity level, which is already overburdened (see the discussion in section 4.3.6).

Once all the primitives have been input, external activation stops. Each grid unit can stay fully active for some number of steps (twelve, currently) before entering an exhausted state. An exhausted unit has a very low level of activation. The properties

Watching the network during recognition trials is interesting. Often the wrong concept has a lead in the activation after only part of the object has been attended to, but as more primitives are "seen," the correct concept wins.

4.3.4 Learning

The dynamic behavior of the network as described in the previous section complicates learning considerably. The network starts with no knowledge of concepts or primitive objects. The properties and relations are built in, but none of the conjunctive or concept units have a meaningful support. During training, an object is input as described above and the simulation runs long enough for each primitive part to be focused upon a few times. Then the correct concept unit receives external feedback strong enough to make it the winner. If the classification made by the network is wrong, a global "surprise" signal is generated and induces learning, which does not occur otherwise (cf. chapter 3).

Learning takes place at the concept units and at the conjunctive units. Having received feedback, the concept units must adjust the weights from the conjunctive units. This is relatively straight-forward. An active concept unit increases the weight on its active input links and decreases the weight on inactive input links. Inactive concept units only learn if they are wrongly thought to be the correct classification. They enter a special state if they are strongly active before receiving inhibition from another concept unit. This is a variation of the delta rule where the weight change is proportional to the difference between the desired output and the actual output.

The difficult learning occurs at the conjunctive units. They receive no direct feedback, and must contend with dynamic activity. Most well-known weight-change algorithms will not work in such conditions. The problem lies in coordinating the reinforcement with the shifting of attention. If specific error information is a direct function of the input, then it will not be correct for all separately-focused-upon subobjects. With back propagation, a separate cycle of error propagation would be necessary for each primitive. But this alone would not be sufficient. The separate phases would need to be coordinated so as to avoid undoing each other's changes.

A simple success/failure signal avoids these problems. Its value remains constant as the focus of attention changes. Not all reinforcement learning algorithms are appropriate, however. A_{rp} [Barto and Anandan, 1985] uses just a success/failure signal, but it maintains histories of activation on the links, and uses these to guide learning. This is ideal for the output layer, but not for the hidden layer, which must separate the subjects into separate representations. Learning based upon recent activity is exactly what should not happen. Input active recently, but quiet now, is the responsibility of other hidden units.

The enhanced competitive learning described in the previous section was used for learning at the hidden layer. As the focus of attention switches, different hidden units become active. Learning occurs at the units currently active, based upon the current input only, without affecting recent learning. Additional forces are needed to guide the competitive learning to a solution. These will be described in this chapter. In order to distinguish hidden units used in this thesis from standard competitive learning units, the term "conjunctive" will be used to describe my modified competitive units.

If the conjunctive units just run a competitive learning algorithm, the performance

bound to the exhausted unit fall to the same level of activation since their only source of activity is the grid unit. The strongest of the other non-zero grid units is no longer inhibited and becomes fully active and has its turn, activating its bound properties as well. In this way, the primitives are attended to cyclically. When there are more than two primitives, some care must be taken to insure that no primitive is locked out. An active grid unit has a potential of 800. When it becomes exhausted, the potential falls to 100, then slowly grows until it is about 200. Suppose there are three grid units and grid unit one is active. It becomes exhausted and grid unit two becomes active. When grid unit two becomes exhausted, grid unit one will still be recovering from its recent activity so grid unit three will get its turn. The grid units will keep going forever, taking turns being active. I have not addressed the problem of when to stop processing altogether; the simulation is simply stopped after some number of steps.

So far, I have described the input of a structured object and the periodic shift of attention from one primitive to another. Assuming the concept has been well learned, I will now describe the recognition of the object. When the first (e.g., large square) grid unit activates, the relation detector between it and the second grid unit also activates and in turn activates the *left-of* relation. The grid unit itself activates *large* and *square*. A conjunctive unit with strong links from just these three units: *large*, *square* and *left-of* has previously been recruited. Conjunctive units respond according to the percentage of their total possible support which is active, so this unit is strongly active. The conjunctive units are mutually inhibitory; no other conjunctive unit will be active. (This mutual inhibition is necessary for the learning algorithm used.) The conjunctive unit has a strong link to the appropriate concept unit since it represents one of the primitives of that concept. Similarly, when the second grid unit activates, there is a conjunctive unit which responds strongly to *right-of* and *circle* and which also has a strong link to the third concept unit.

One difficulty that arises is making the concept unit response proportional to the sum of activation of both conjunctive units (both are necessary parts of the definition) even though they are never simultaneously active. The solution used gives links to concept units a memory: any link which has been active recently will provide activation as though it still were.[3] When the first conjunctive unit of our example turns on, the concept unit receives some activation (about half of what it expects). When the second turns on, it will be receiving activation both from the second conjunctive unit and remembered activation from the first, even though it is now off. This strongly activates the concept unit; the input has been classified.

Making remembered values work is a little tricky. The remembered value of a link at time t, m_t, is the following, where o_t is the output of the conjunctive unit which is the source of the link:

$$m_t = m_{t-1} + \begin{cases} \frac{(o_t - m_{t-1})}{8} & \text{if } o_t > m_{t-1} \\ \frac{(o_t - m_{t-1})}{50} & \text{if } o_t < m_{t-1} \end{cases}$$

The remembered value moves towards each new output value. Positive changes are several times larger than negative changes, because a primitive subpart will be off more than on if there are several other subparts. The above ratio (8/50) was arrived at experimentally working with two and three part objects.

[3] Section 4.3.6 points out some of the weaknesses of this approach and discusses alternatives.

is poor. Since they receive no feedback, the task to them is identical to a series of unstructured inputs corresponding to the primitive subobjects. (In this setup, there is nothing special about relation nodes; they are just another input.) It is not surprising that the conjunctive units do not partition the input correctly. For example, it may be important to separate (*above* ∧ *small* ∧ *square*) and (*above* ∧ *small* ∧ *circle*) while grouping together (*left-of* ∧ *small* ∧ *circle*) and (*left-of* ∧ *large* ∧ *square*), depending on the specificity of the definitions to be learned. Unsupervised learning cannot be expected to do this spontaneously.

One manifestation of this problem is the simultaneous recruitment of the same conjunctive unit for different primitives. In order to help the conjunctive units distinguish primitive objects which are part of different concepts, top-down links from concept to conjunctive units have been added which learn according a bounded Hebbian rule, as described in section 3.5. This helps, except with primitives which are part of the same concept. Intra-concept primitives are made more similar because they now have an additional input in common: the top-down feedback. It is crucial that the learned description of a three-part object have three parts. There is nothing in the network as described to force or even encourage this. As the simulation ran with such a network, separate parts of the same object tried to recruit the same conjunctive node, the support for which resembled a confused combination of them all. Some way to force a three part object to have a three part description is necessary.

Since the grid units control the change of attention for the input, they can do the same for the conjunctive nodes. Dynamic links from grid units to conjunctive units were added with a fast weight change very much like that from grid units to properties. Once a conjunctive unit binds to a grid unit, it does not become active unless the grid unit does. A bound conjunctive unit can represent only one primitive for any input no matter how similar others may be. A conjunctive unit binds to the active grid unit only after it wins the competition among grid units. This is important. When there is first a change of attention, several conjunctive units will be active to some extent. If binding to the grid unit were based upon mutual activity, far too many would commit to the same primitive.

Now the conjunctive units are encouraged to distinguish between concepts and are forced to distinguish between the primitives of a single concept. This allows fine distinctions to be made. Running the network reveals a problem with the top-down feedback, however. Stated in general terms, the trouble is that an incorrect early hypothesis can bias further observations which would otherwise confirm the correct hypothesis. More specifically, once a concept unit dominates the others, it tends to continue to do so, no matter what further evidence arrives. Suppose an instance of concept one is input. After focusing on the first primitive, concept two has slightly more input than concept one. This could happen for any number of reasons, including identical primitives for each. Because the concepts are mutually inhibitory, concept two stays active and concept one becomes idle. When the second primitive activates, it matches a concept one conjunctive unit well, but a concept two conjunctive unit wins the competition simply because the feedback from the active concept two node is exciting it and inhibiting the concept one primitive. This makes concept two even stronger and the problem perpetuates.

The problem goes away if the top-down feedback is reduced, but if it is reduced too much, the confusion between primitives of separate objects resumes. I was able to

achieve fairly good performance by experimentally setting the top-down feedback to a balanced level and by relaxing the competition between conjunctive and concept units so that losers would stay on a while before becoming inactive. After this change, incorrect top-down feedback no longer totally suppresses activation on inconsistent conjunctive units. Evidence for the correct but suppressed concept unit can accumulate enough to override an incorrect head start.

4.3.5 Experimental Results

The experiments reported in this section were run by a driver program which inputs an appropriate object, runs some number of simulation steps, then provides feedback and runs some more simulation steps. The concept definitions are specified using a simple input language which gives instructions for building an object of the appropriate type. For simplicity, instead of giving meaningful labels to the values, they are numbered, so there is shape 0 and shape 1 instead of square and circle. The concepts are simply numbered as well. Below is a sample specification for concept number 3.

3 # size 0 shape 0 % right # size −1 shape 3 %

The "#" and "%" delimit attribute/value lists, which describe a primitive. In between the attribute/value lists are directions for moving to the location of the next primitive. The sample definition means, to make an object of type 3, start with a primitive of size 0 and shape 0, then move right and make a primitive with randomly chosen size (a value of −1 means "don't care") and shape 3. As described above, the primitive objects are entered one at a time by activating the appropriate grid unit and property units.

Several experiments were run in a setup having a 4 × 4 grid, two attributes with 4 values each, which I will call "shape" and "size," 15 conjunctive units and 4 concept units. Having more conjunctive units than necessary provides a stricter test as competitive learning can be unstable in such cases (see chapter 3). For the first experiment, the concept definitions used were the following:

0 # shape 3 size 3 % down # shape −1 size 0 % right # shape 0 size −1 %
1 # shape 2 size 2 % down # shape 0 size 3 % left # shape 1 size 0 %
2 # shape 1 size −1 % down # shape 0 size 2 %
3 # shape 3 size 2 % right # shape 2 size 1 %

Concepts 1 and 3 are exactly specified and will appear the same each time. Concepts 0 and 2 will have more variation: the −1 values will be replaced by randomly selected values during the simulation. Concepts 0 and 1 have 3 primitives each; concepts 2 and 3 have 2 primitives each.

The grid units were set to stay active for 10 simulation steps before entering the exhausted state. Each round of trials included one example of each concept. After sequentially entering each primitive object, 80 simulation steps were run, then the correct concept received feedback and 80 more steps were run. After four passes, the concepts were well learned. Figure 24 is a screen dump from the simulator display showing the state of the network. The display is fairly complex. There are icons in three shapes:

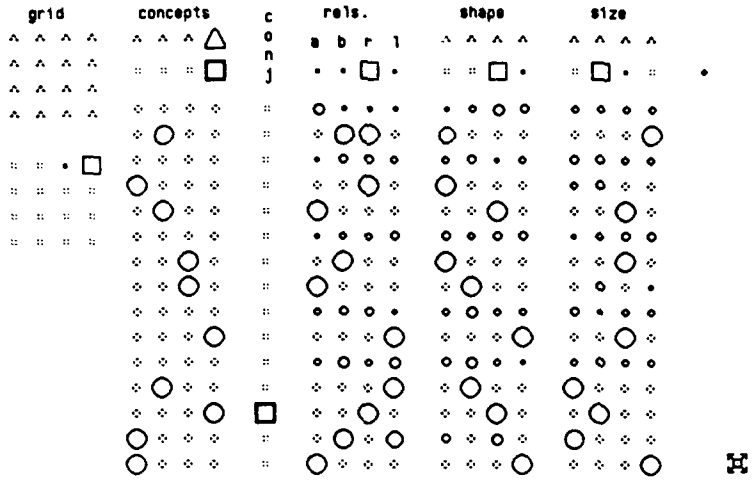

Figure 24: Network after four passes, processing an example.

triangles, squares and circles. The squares represent internal unit potentials. The size of the square is proportional to the activity of the unit. The triangles represent external unit potentials used to input data and are displayed above the internal units they activate. On the left is the grid array; above it are the external grid units. All but two grid units are quiet: the current input has two primitive parts. Grid unit 3 is being attended to.[4] Grid unit 2 has a very low level of activity.

Along the top to the right of the grid are 4 square concept units. Above them are the corresponding triangular external concept units. External concept unit 3 is active, which means feedback has been provided for this concept. The corresponding internal concept unit is also active. To the right of the concept units are the four relation units, with only right-of (labeled "r") being active. This is active because the primitive being attended to is to the right of the other input primitive. *These units do not receive* external activation, but are activated by relation detection units which are not shown in the display. Next are the shape and size units. Shape 2 and size 1 are active. They were originally input externally, but are now being driven by the grid unit to which they are dynamically bound. The column of square icons labeled "conj" represents the activity of the conjunctive units. One of these is active; it is responding to the current primitive.

The circular icons represent weights on links between displayed units. There are four long, rectangular areas of circular units. The left-most matrix of circular icons shows weights from the conjunctive units to the concept units. To find the weight from conjunctive unit 3 to concept unit 0, look at the icon in row 3, column 0. It is large, meaning that link has a strong weight. Notice that these links are all either very strong or zero at this stage. The other matrices of circular icons show weights from the relations and properties to the conjunctive units. *Conjunctive unit 3 responds to* {shape=0 $\wedge$ right-of} with no strong commitment to any size. This exactly fits one of

[4] All numbering begins with 0.

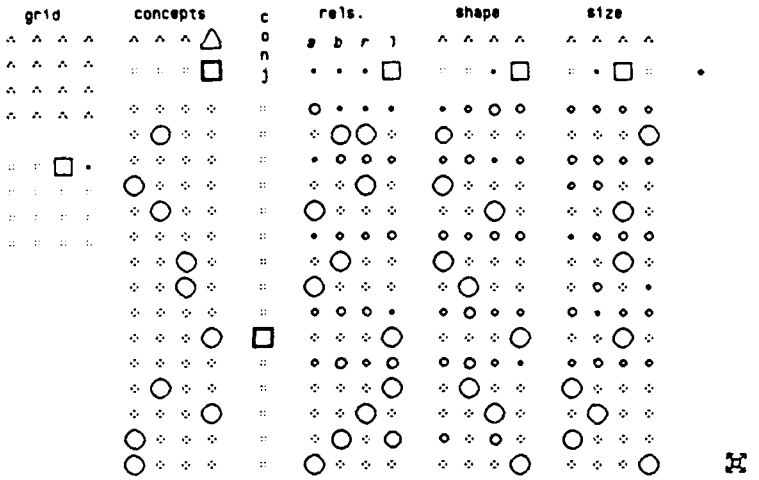

Figure 25: Same as previous figure, a few steps later.

the primitives of concept 0, which explains the strong weight there. Figure 25 shows the same network a few steps later after attention has shifted to the other input primitive.

In order to push the network, a more complex set of definitions was tried:

0 # xshape −1 xsize 3 % down # xshape −1 xsize 0 % right # xshape 0 xsize −1 %
1 # xshape 2 xsize −1 % down # xshape −1 xsize 3 % left # xshape 1 xsize −1 %
2 # xshape 1 xsize −1 % down # xshape 0 xsize −1 %
3 # xshape −1 xsize 2 % right # xshape −1 xsize 1 %

These definitions have more don't care conditions than the first set. This exacerbated the problem of incorrect top-down feedback due to a false first guess suppressing the *correct* conjunctive units. An adjustment was made by lowering the amount of inhibition between the conjunctive units, so they would be more likely to be simultaneously active. This provides more information to the concept units—the non-winners can indicate how strong they are, and this often allows the correct concept to predominate after a couple of cycles of attention when it would have stayed suppressed before. This lowered competition inhibits the early selection of conjunctive units in the initial trials since several often remain active. This is countered by increasing mutual inhibition in case of surprise. After receiving feedback, a clear winner is necessary. These adjustments were successful.

The next experiment combined four-part objects and two-part objects:

0 # xshape 3 xsize 3 % down # xshape −1 xsize 0 % right
  # xshape 0 xsize −1 % up # xshape 2 xsize 2 %
1 # xshape 2 xsize 2 % up # xshape 0 xsize 3 % left
  # xshape 1 xsize 0 % down # xshape 2 xsize 3 %
2 # xshape 1 xsize −1 % down # xshape 0 xsize 2 %
3 # xshape 3 xsize 2 % right # xshape 2 xsize 1 %

In the previous experiments, the number of trials per structured object was the same for all objects. This data makes that infeasible, as the four-part objects require twice as

Figure 26: Two structures with an equivalent description.

much processing time as the two-part objects. Too much time on the two-part objects overlearns the specific input. Not enough time on the four-part objects causes failure to adequately learn all parts. This was corrected by varying the length of a trial with the size of an object. Clearly, automatic means of controlling the amount of attention given an object and the amount of learning for any one trial are desirable. There does not seem to be an principle difficulty with this, though it not been done. After making the timing adjustments, the above definitions were learned correctly.

4.3.6 Discussion

Many of the problems encountered with this implementation are interesting and of general significance. The representation of relations and concepts involving relations is an important problem. In the above implementation, relation roles were treated as just another property when learning object primitives, so that a square to the left of a circle had two components: (square $\wedge$ left-of) and (circle $\wedge$ right-of). This representation is simply inadequate. Completely different structures will look the same in such a representation; figure 26 gives an example.

Some way to link specific primitives together is needed, so that instead of having a circle left of *something* we can specify exactly what the circle is left of. Combining relations and properties to form the conjunctive representation of primitives also leads to a loss of generality. If structure and primitive descriptions were separate, then it would be easy to reason about them separately: to see the same structure involving different primitives, to recognize the same primitive in different structures. Formulating a structured description as primitives with relational glue is more satisfactory. This was not attempted in this preliminary implementation because it is considerably more complex than the approach taken. Such a solution would probably involve replacing the concept units used here with several units, at least one per primitive plus one for the whole object. These would be distinct from the conjunctive units insofar as their state would represent what has been recognized so far about the object; the conjunctive units compete to represent the primitive currently being attended to.

The basic trouble with having a single concept unit is the paucity of state information which can be recorded in a single potential. Does an activity level 75% of the maximum mean 75% of the object's primitives are there, or that all the primitives are present, but with average confidence of on 75%? Overburdening the information contained in a unit's potential is common in connectionist models. Does a low level of activity mean weak evidence or non-relevance? In a winner-take-all network it is hard to have a fair competition without letting the units reach strong potentials, but these strong potentials can start activity, cause dynamic bindings, etc. in the rest of the network prematurely, before there is a winner. It might be possible to gate the winner-take-all networks so that no activity leaves until there is a winner, but this hinders the ability of the network to use feedback from different levels to pick the correct winner. It would

much easier to design networks if units had several output values: strength of evidence for the represented hypothesis, recent activity, correlation with reward and punishment, relevance of context, willingness to bind, etc.

Switching attention without confusing the pre- and post-switch contexts is another problem of general significance. The particular manifestation of this problem encountered is the need to prevent dynamic binding of the old grid unit with the new properties. The solution is to have newly attended to objects be especially active and to require this extra activity for binding. The solution is not seem optimal. It is yet another problem foisted on potentials (see the discussion above). Switching of attention is likely to be a general problem in connectionist networks because of the lack of a central interpreter. The distribution of control in connectionist networks makes it difficult to switch attention simultaneously throughout the network. Parts of the previous context should remain, while other parts are replaced. Exactly what to attend to next should in general be influenced by what is being attended to now, so the old context must stay active long enough to get the new context going. (Notice the resemblance to the "frame problem" in AI.) The binding problem encountered here is just the tip of the attention iceberg. This model does not address conscious attention or awareness; it says nothing about which structured object to process, only what happens during that processing.

Contextual control over attention was not attempted, partly because there is so little context in the problem used. If top-down priming were used, it would probably come via the conjunctive units which are in a winner-take-all network. The mutual inhibition of winner-take-all networks makes it impossible to activate candidates for the next cycle of attention while the current conjunctive winner is suppressing everything. Placing the conjunctive units in a winner-take-all network has several advantages and disadvantages. It is necessary for the learning algorithm, which uses a simple approach to credit assignment: if a conjunctive unit wins the competition, it gets all the credit (for that primitive). If several conjunctive units were active, the credit would be too widely distributed. Allowing less strong conjunctive units to stay active at lower levels would, on the other hand, allow the all the concepts to receive input proportional to the evidence supporting them. The suppression of evidence from non-winners causes the problem, discussed in section 4.3.3, of correct concepts being suppressed when they otherwise would not because top-down feedback from an incorrectly hypothesized concept unit prevents otherwise strong conjunctive units from winning. Improved performance resulted from an ad hoc compromise between winner-take-all and no competition achieved by having the defeated conjunctive units fade slowly so that they would be active for a bit longer than they otherwise would be. This solution is ad hoc because the balance is optimized for this particular network. Correct behavior and efficient implementation of winner-take-all networks is a tricky and important problem (see [Chun *et al.*, 1987]).

Another basic problem encountered in this implementation is the temporal summation of evidence. The concept units in this model cannot simply sum incoming activity since the units representing the component primitives are never simultaneously active. This problem is fundamental because sequential arrival of evidence is unavoidable, even if it is not necessary for structured objects as hypothesized here. Simply storing previous evidence as an increase in the potential of the unit, so that each piece of evidence simply increments the concept unit's potential, is not an adequate solution. In the network described here, this technique would be unable to distinguish new evidence from

repetitions of old evidence, and it is not possible to focus attention on each primitive only once, as described below. In this implementation, the requisite additional state information is kept on the individual links. A concept can fully activate only if all required links have been recently active.

Making the link memory work correctly presents some difficulties. Many of these issues remain even if link memory is replaced by some other mechanism, so they meri. discussion. One of the problems was mentioned above. While the conjunctive units are still competing to find a winner, many of them are active; the links to concept units are recording this activity. If the links simply remembered the strongest of their recent inputs, the losers would be only slightly weaker than the winners. The remembered values must grow slowly when the input is active and decrease even more slowly when it is not. Some considerations influencing the rate of change are:

- The increase in the remembered value should be fast enough so that after one round of attention to each subpart, the concept receives strong support. This means that the increase is closely tied to the number of simulation steps before grid units become exhausted.
- The increase should not be so fast that conjunctive values which fired during competition, i.e., before there was a clear winner, are remembered. What is remembered is strongly influenced by the number of steps per round of attention. If three steps of wide-spread activation are required to find a conjunctive winner and attention stays focused for just 1 more step, a decaying average of recent activity will barely distinguish the winner from near winners. In the networks tested, which had a very small number of possible input attributes, many conjunctive units would respond fairly strongly to each primitive (this problem is compounded in the early stages of learning). It is not desirable for the weights on links from all of these to be incremented. Since the weight increment is proportional to an average of recent activity, the winner needs to stay active by itself for a while to up its average and let the others decay.
- The decrease should be fast enough to clean up accidentally remembered values due to competition, but slow enough to let truly remembered values stay high while all the other subparts are getting a turn. The more steps per cycle of attention, the slower this decay must be.
- The decrease must be fast enough so that when some entirely new object is considered, residual activation of the old concept is effectively gone. Currently, this is not a problem as the remembered values are reset to zero manually between trials. In more complex computations, this problem will require a solution.

The ratio between growth rate and decay rate used is 50/8. This is totally ad hoc and was experimentally selected. No fixed rate will be adequate as the size, complexity and familiarity of the object being classified greatly alter the requirements.

The winner-take-all behavior of the conjunctive and concept units, combined with the top-down biasing of conjunctive units necessitates more than one attentive pass over the primitives of a subobject. In general terms, the first pass may be misinterpreted, but enough information may get through so that on the second pass the primitives and the object are correctly classified. The need for more than one pass is especially true

during the learning phase. If a concept is originally misclassified, then the primitives are probably misclassified as well (i.e., the conjunctive units which are responding are not those which will eventually represent those primitives). After feedback is given, the correct concept unit changes the weights on incoming links according to their recent activity. But if the concept had been misclassified, the recently active conjunctive units are the wrong ones. This results from the strong top-down influence of the concept units on the conjunctive units. After the correct concept unit begins providing the top-down feedback, the correct conjunctive units will begin responding, but it takes many simulation steps for them all to respond. Until they do, the weights on links from them are decreased while the weights on links from incorrect conjunctive units are increased. These wrong steps will be undone during further cycles of attention on the primitives, so each primitive must be focused upon several times.

Most of these difficulties are especially acute before the network has learned. Because of the nature of the competitive learning algorithm which underlies the recruitment of conjunctive units, conjunctive units must respond strongly even before they have found a clear role to play. The result is that in the early stages of learning, lots of "confused" conjunctive units are competing to represent several different primitives. Most of the additional mechanisms described are designed to lock conjunctive units into some particular role early so they will have a chance to learn exactly what that role is (i.e., to learn exactly which combination of features and relations is necessary).

Another problem with conjunctive units is the requirement that each be connected to all property units. No total interconnection design can work for large scale problems. The same is true of the fast binding links from the grid units. Clearly, single units must be replaced by expanding networks of units. These problems are not insurmountable but neither are they trivial. Feldman [1982] investigates some techniques for efficient binding.

The network described above cannot run top-down. If a concept is activated, the correct description will not in turn activate. The trouble involves separating the parts of the description. During training, each complex object is input one primitive at a time. In order for top-down activation to work correctly, the primitives must be separated and bound to grid units automatically. This seems possible in theory, but requires further developments so that grid units can be activated internally in a consistent manner. This would be an interesting development because it would demonstrate internal guidance of the attention mechanism. Currently, the experimenter enters all the pieces to be focused upon explicitly.

Improving the performance of the network described above is important. Many simplifying assumptions were made which are not realistic. However, it would be a mistake to try to optimize structured concept learning in isolation. Concept learning is really only useful in a richer context. Extending the scope and functionality of the network further constrains the processing and changes the solutions already devised. Some interesting extensions include:

- Extending the network to handle general relations, e.g., "John loves Mary." The position grid evolves into a kind of working control space with no semantics (i.e., having nothing to do with location) or perhaps with general purpose semantics, such as case roles like "subject." I like to think of this as roughly analogous

to STM. The control space is highly connected, dynamic, and can hold things together for further processing. Because of this power, it is resource intensive and can hold only a few items.

- Storing both general concept definitions and specific facts about individuals.
- Letting complex concepts themselves be subobjects in even more complicated concepts. Hierarchies of concepts are very important to the eventual success of the paradigm. With the control outlines given above, such a mechanism is necessary to allow complex objects to be treated as primitive during computation. Once a complex object has its own handle, it can bind to a control (grid) unit and cycle on and off as attention shifts. Only properties of the object as a whole will come on, preventing cross talk between the subparts. This explanation is oversimplified. Actually making such a mechanism work presents many challenges.
- Developing a rule-like mechanism so that relations can be inferred from the presence of other relations. This is discussed further in chapter 5.
- Enriching the relation space so that relations have properties, etc.
- Adding type hierarchies to the relations and objects so that properties can be inferred.

Eventually connectionist theories must address problems too hard to be solved in one pass through a network or by a single relaxation. This work is a step in that direction. It is important to note that although there is a sequential component to the recognition process, at each step a typical connectionist computation occurs: it is parallel and error tolerant; it generalizes, allows a top-down and bottom-up flow of information. etc.

Chapter 5

SUMMARY AND FUTURE WORK

5.1 Summary

Chapter 2 describes a fast, exact implementation of a parsing network. A network can be easily built for any context-free grammar, and parses in $O(n)$ parallel steps, where n is the length of the input. For any given network, the length of input which can be parsed is strictly limited. The size of the network is $O(n^3)$ for grammars in Chomsky normal form—having at most two nonterminals on the right hand side of a production. The network size grows exponentially with longer productions. The network's behavior is exact and non-evidential. The strategy used is to build the entire chart for a bottom-up chart parser, with a unit for each nonterminal at each location in the chart. The collective activation of nonterminals in the correct position to satisfy a production will activate the nonterminal node from the left-hand side of that production in the correct location. A second, top-down pass of activity beginning at the root node isolates those units actually participating in a complete parse.

Some modifications where made to explore the network's ability to tolerate less exact input. By lowering the threshold on units recognizing a satisfied production, it is possible to complete a parse with a missing component. The lowered activity of the units representing the parse indicates the incompleteness. This modification by no means captures the full range of "near-miss" parses. The parts which are present must be in the same location as with a complete parse, so missing words which cause subsequent words to be shifted one place throw the entire parse off.

Because the entire chart for the grammar is expanded a priori, with each production being replicated many times in different positions, learning is very difficult. A very involved mechanism was implemented which can learn new productions in limited circumstances. The first step was achieving local, isolated learning of a new production involving existing nonterminals. The new production is learned when there is a near-miss parse and a bottom-up parse of the gap which is one production short of tying into the surrounding tree. A special unit coexisting with each non-terminal unit detects this condition and stimulates an uncommitted "match" unit to recognize that production. The match unit knows which production to recruit because there is only

one possibility—or learning would not proceed.

So far the process is only a little complex. But for the network to be consistent, the same production must be recognized everywhere in the network (i.e., not for a single set of starting positions and lengths of the components). This is achieved by first notifying a central location of the new production, and then broadcasting it throughout the network. Because there is no symbolic communication in a connectionist network, the notification must result from something being active. In this case, an especially high level of activity on nonterminal units stimulates corresponding units in the central template. Because a single unit may be acting as either parent, left-son or right-son (the learning was only implemented for Chomsky normal form grammars) and the central template must know which in order to correctly interpret the incoming activity, each nonterminal unit of the original network is replaced by three units, one for each possible role. Each is connected to the template unit filling the same role, so hyperactivity can be correctly interpreted. A similar process works in reverse to communicate the new production globally. The central template units representing the production become hyperactive and cause, throughout the network, patterns of activity which induce local learning of the new production.

Chapter 3 explores feature-based concept learning. The bulk of the chapter concerns a new hidden unit learning rule which uses a "surprise" or failure signal to induce recruitment of hidden units representing conjunctions of input features. If the occurrence of conjunctions of inputs is important, then until a representation is recruited there will be consistent failure of the network while those inputs are active. If recruitment of hidden representations occurs for inputs active during surprise, then the important conjunctions should recruit a representation. The advantages of the technique include the simple feedback from the output units to the hidden units and speed. Only a few trials are necessary to learn many concepts. In order to make the learning easier and because sparse connectivity is necessary to control redundant learning, the hidden units are designed to represent *pairs* of inputs. When the concept definitions consist of a pair of inputs, the network works well. For longer definitions, high-level pair units which represent pairs of pairs of inputs are used. The surprise induced recruitment of these units does not work well. On the whole, the technique is inadequate.

Because hierarchies of pairs worked so poorly for representing long conjunctions, another approach was tried. In order to have a single unit represent large conjunctions, it is necessary to have dense connectivity. To prevent similarly connected units from duplicating each other, it helps to have mutual inhibition, letting them compete for the right to represent each input. This is similar to competitive learning, but must operate as the hidden layer in a supervised learning task. Competitive learning is unsupervised. In order to make the competitive learning units sensitive to the categorization imposed by the instructor, links were added from the output layer to the hidden layer of competitive units. These links are weighted strongly, so that they have more influence than the individual bottom-up links. Very similar category definitions will still be confused, but the top-down links improve performance significantly.

The pair-recruitment work absorbed many months of work, and perhaps hundreds of experimental variations before being abandoned. The enhanced competitive learning was relatively easy to develop and works better. Even so, it is not a general as other techniques, notably back propagation. The link-specific error signals used in back

propagation make it quite powerful. However, back propagation seems to be limited to moderately-sized feed-forward networks. It is inappropriate for the dynamic environment used for structured learning (chapter 4). The competitive learning method fits in well.

Chapter 3 also reports experiments with the back propagation learning technique. Various architectures were used for the same problem in order to explore their effect on the solution found by back propagation. In particular, the ability of the network to generalize the solution to previously unseen inputs was tested. Biasing the solution with a priori structure did enhance the desired generalization. The fact that an unstructured network made the generalization for a large number of inputs was surprising. The question remains, how to predict and control the solution found by back propagation.

Chapter 4 addresses the problem of representing and learning structured objects. A simple object is one which can be described by a set of features. A structured object is composed of two or more simple objects which are interrelated. The first issue to address is the problem of dealing with several objects in a connectionist network without crosstalk. Rather than use multiple representation spaces, I argue the necessity of time sharing within a single space. In order to process a structured object, attention must switch between the simple component objects in turn. When an object is active, so are its relations, so that if a square is left of a circle, focusing attention on the square also activates "left-of." As attention is focused upon the primitives in turn, evidence must accumulate over time at the concept nodes. A preliminary implementation of these ideas uses link memory so that all recently active primitives provide evidence simultaneously. The network succeeds in learning descriptions of simple structured objects. The abstraction to a single representation space makes the learning naturally position independent. The features of each primitive input dynamically bind to a grid unit representing their location. From that point, they are dependent on the grid unit for activation. The grid units are mutually inhibitory, so only one is active at any time. Active grid units soon decay, and the attention shifts to the next strongest grid unit. Modified competitive learning units are used to recognize the primitives. Many interesting problems arose during the implementation and are discussed in chapter 4. Much work needs to be done, both improving the solutions used here and extending them to more complex tasks. A sketch of how the representation would help in the implementation of inferences is provided in the next section.

Not reported in this thesis, but an integral part of the research, was work on the Rochester Connectionist Simulator [Feldman *et al.*, 1988], including a parallel implementation [Fanty, 1986a]. The simulator has since been extended by Goddard and Lynne [Goddard *et al.*, 1987].

5.2 Rules

Some connectionist inferences are fast, almost automatic: to get $A \rightarrow B$, add excitatory links from the representation of A to the representation of B. This may mean a single link from the A node to the B node, or a more complex pattern of increased weights if A and B have distributed representations. However, rules like $P(x) \wedge Q(x,y) \rightarrow R(y)$, are not be so easily realized. It is not sufficient to have P and Q excite R, as this does not

distinguish R(x) from R(y). It is necessary to associate R with whatever fills the y role. This is an instance of the binding problem [Shastri and Feldman, 1984, Smolensky, 1987]. The mechanism described in chapter 4 addresses this problem. Because the objects x and y are being considered sequentially, binding can result from simultaneous activation without cross talk.

Some work has been done on connectionist rule interpretation. Parsing must deal with grammatical rules, so a parsing network such as that in chapter 2 as well as others [Selman and Hirst, 1985, Cottrell, 1985] are a sort of rule interpreter. The binding problem arises in this context insofar as a rule such as S $\rightarrow$ NP VP should only fire if the NP and VP represent contiguous segments of the input. This is insured in the network described in chapter 2 by explicitly replicating the rules for each possible binding. As a general implementation of rules, this is a nonsolution.

Toureksky and Hinton [1985] have implemented a simple rule interpreter, complete with variable binding. The work is preliminary, serving as an existence proof to quiet critics who claim distributed connectionism cannot do such things. The domain is simple. Statements consist of triples of letters. In order to achieve rule matching, the entire representation space is replicated three times. Variables are limited to the first position of the triples. Rules are explicitly hard-wired in; no learning occurs. Probably the most serious limitation of the work is the way in which the space of all possible letter triples is represented. This does not promise to scale to rules involving semantically complex entities.

Ballard and Hayes [1984] describe a connectionist inference mechanism based on resolution. They make a number of important assumptions: each clause may be used only once; the knowledge base must be logically consistent; and a large, complex network linking the clauses and all the possible unifications must be prewired. This latter assumption, especially, calls into question the networks appropriateness more general purpose inferencing in a huge, constantly changing, internally inconsistent (if it is human-like) intelligent network.

Figure 27 sketches the major ideas behind my proposed treatment of rules. Two people, Mary and John, are represented by a single unit each. That Mary is innocent is depicted by a link from the Mary unit to the innocent unit; whenever the Mary unit is active, so is the innocent unit. Likewise, John is a scoundrel. Two-place relations are represented by three units: one for the subject, one for the object and one to tie them together. A representation for *loves* and *hurts* is depicted. Unlike properties, facts involving multi-place relations cannot be instantiated with a single link. Mary may love several people. Simply inserting a link from the Mary node to the $\overrightarrow{\text{loves}}$ node would not indicate whom Mary loves. In figure 27, the fact that Mary loves John is represented by two triangle-shaped units. When attention is focused on Mary, innocent and $\overrightarrow{\text{loves}}$ receive some activation. If, given the current context, loves is important, then $\overleftarrow{\text{loves}}$ will receive enough activation to prime John and attention will switch from Mary through loves to John. "Mary loves John." The key for controlling this association from Mary to John is the link between the two triangle units representing the specific fact that Mary loves John.

What is also shown is the rule, "if an innocent loves a scoundrel, then the scoundrel will hurt the innocent." The rule is represented by five numbered units. Units 1 and 2 are the two preconditions for the rule. Taken together, they and the units connected to

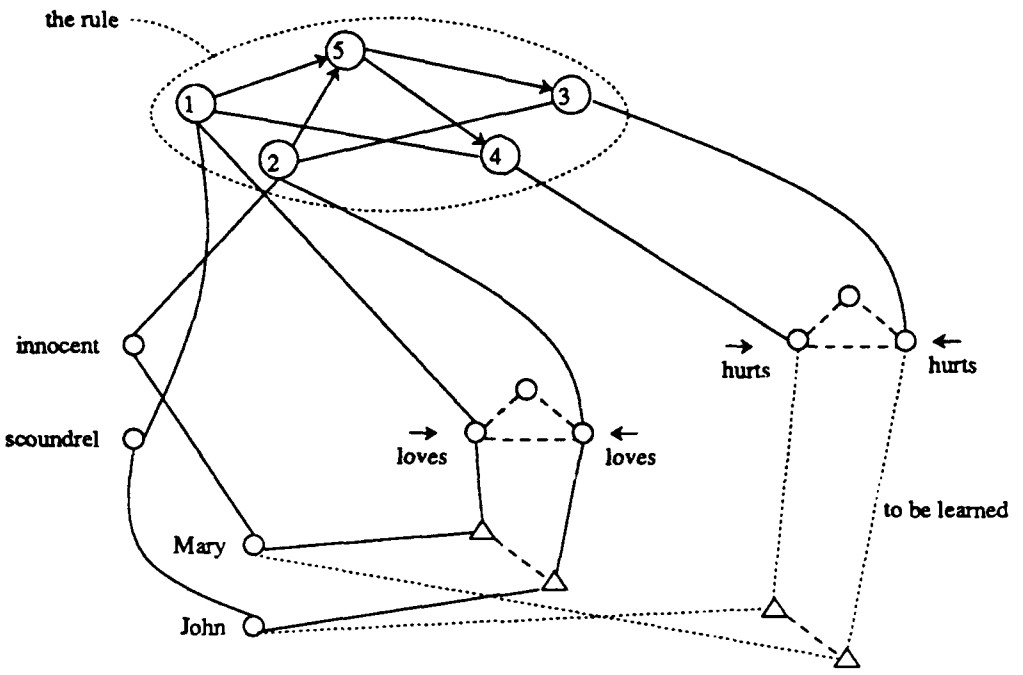

Figure 27: A connectionist rule.

them represent the situation where an innocent loves a scoundrel. Unit 5 will activate only if the two preconditions are met. So far, this is very much like the structured object recognition described in chapter 4. Inferencing occurs after unit 5 becomes active indicating recognition of the preconditions of the rule.

The action part of the rule must instantiate the fact that John will hurt Mary. The network continues to switch attention back and forth between John and Mary. When Mary is active, so is the unit representing the second condition of the rule. This unit activates action unit 3 which activates $\overleftarrow{\text{hurts}}$. Now Mary and $\overleftarrow{\text{hurts}}$ are simultaneously active: Mary is hurt. Similarly when John is active the first condition of the rule activates action unit 4 which activates $\overrightarrow{\text{hurts}}$: John is hurting someone. This inference could be made permanent if new units are recruited to record the relation (shown with dotted lines in figure 27), or the "thought" could be fleeting. How to control whether the inference is made permanent is a separate problem.

Because of the temporal binding mechanism, at most one rule could fire at any time, though several could compete for the right to fire. It is conceivable that all rules participate in a giant winner-take-all network, but some additional control is probably necessary. Rule search could be facilitated by backwards priming, so that, in the above example, wondering why Mary looks hurt could focus our attention on her loving John the scoundrel.

The rule representation outlined here is very similar to the structured object representation explained in chapter 4. A similar learning mechanism could be used to actually learn rules. Given a situation such as *John hurts Mary*, and a desire to learn why, a learning process could be initiated with *x hurts y* automatically linked in as the action part and with the condition part as the concept to be learned. That a mechanism

chosen strictly for its structured learning capabilities also promises to work for rules and rule learning is very encouraging.

Bibliography

[Aho *et al.*, 1974] A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *The Design and Analysis of Computer Algorithms.* Addison-Wesley, Reading, MASS, 1974.

[Anderson, 1985] James A. Anderson. Synaptic modification. In William B. Levy, James A. Anderson, and Stephen Lehmkuhle, editors, *Synaptic Modification, Neuron Selectivity, and Nervous System Organization*, pages 153–173, Lawrence Erlbaum Associates, Hillsdale, New Jersey, 1985.

[Ballard, 1986] Dana Ballard. Cortical connections and parallel processing: structure and function. *Behavioral and Brain Sciences*, 9(1):67–120, 1986.

[Ballard and Hayes, 1984] Dana H. Ballard and P. J. Hayes. Parallel logical inference. In *Program of the Sixth Annual Conference of the Cognitive Science Society*, 1984.

[Barto, 1985] Andrew G. Barto. *Learning by Statistical Cooperation of Self-Interested Neuron-Like Computing Elements.* Technical Report COINS Technical Report 85-11, Computer Science Department, University of Massachusett at Amherst, 1985.

[Barto and Anandan, 1985] Andrew G. Barto and P. Anandan. Pattern recognizing stochasitc learning automata. *IEEE Transactions on Systems, Man, and Cybernetics*, 1985.

[Brunner *et al.*, 1956] Jerome S. Brunner, Jacqueline J. Goodnow, and George A. Austin. *A Study of Thinking.* John Wiley & Sons, Inc., New York, 1956.

[Chun *et al.*, 1987] Hon Wai Chun, Lawrence A. Bookman, and Niki Afshartous. Winner-take-all structure. In *Proceedings of the 10th International Joint Conference on Artificial Intelligence*, pages 380–387, 1987.

[Cohen and Feigenbaum, 1982] Paul R. Cohen and Edward A. Feigenbaum, editors. *The Handbood of Artificail Intelligence.* Volume 3, Willian Kaufmann, Inc., Los Altos, California, 1982.

[Cottrell, 1985] Garrison W. Cottrell. *A Connectionist Approach to Word Sense Disambiguation.* PhD thesis, University of Rochester, 1985.

[Dennett, 1988] Daniel C. Dennett. When philosophers encounter artificial intelligence. *Dædalus*, 117(1):283–295, 1988.

[Elman, 1988] Jeffrey L. Elman. *Finding Structure in Time.* Technical Report CRL 8801, Center for Research in Language, University of California, San Diego, 1988.

[Fanty, 1985] Mark Fanty. *Context-Free Parsing in Connectionist Networks.* Technical Report 174, Computer Science Department, University of Rochester, 1985.

[Fanty, 1986a] Mark Fanty. *A Connectionist Simulator for the BBN Butterfly Multiprocessor.* Technical Report 164, Computer Science Department, University of Rochester, 1986.

[Fanty, 1986b] Mark Fanty. Context-free parsing with connectionist networks. In John S. Denker, editor, *Neural Networks for Computing*, pages 140–145, 1986. AIP Conference Proceedings 151.

[Feldman, 1982] Jerome A. Feldman. Dynamic connections in neural networks. *Biological Cybernetics*, 46:27–39, 1982.

[Feldman, 1986] Jerome A. Feldman. *Neural Representation of Conceptual Knowledge.* Technical Report, Computer Science Department, University of Rochester, 1986.

[Feldman and Ballard, 1983] Jerome A. Feldman and Dana H. Ballard. Connectionist models and their properties. *Cognitive Science*, 6:205–254, 1983.

[Feldman *et al.*, 1988] Jerome A. Feldman, Mark A. Fanty, Nigel Goddard, and Kenton Lynne. Computing with structured connectionist networks. *Communications of the Association for Computing Machinery*, 31:170–187, February 1988.

[Goddard *et al.*, 1987] Nigel Goddard, Kenton Lynne, and Toby Mintz. *The Rochester Connectionist Simulator.* Technical Report 233, Computer Science Department, University of Rochester, 1987.

[Grossberg, 1976a] Stephen Grossberg. Adaptive pattern classification and universal recoding. *Biological Cybernetics*, 23:121–134, 1976.

[Grossberg, 1976b] Stephen Grossberg. On the develpment of feature detectors in the visual cortex with applications to learning and reaction-diffusion systems. *Biological Cybernetics*, 21:145–159, 1976.

[Hayes-Roth and Hayes-Roth, 1977] Barbera Hayes-Roth and Frederick Hayes-Roth. Concept learning and the recognition and classification of examples. *Journal of Verbal Learning and Verbal Behavior*, 16:321–338, 1977.

[Hendler, 1987] James A. Hendler. Marker-passing and microfeatures. In *Proceedings of the 10th International Joint Conference on Artificial Intelligence*, pages 151–153, Morgan Kaufmann Publishers, Inc., Los Altos, CA, 1987.

[Hillis, 1985] W. Daniel Hillis. *The Connection Machine.* The MIT Press, Cambridge, Massachusetts, 1985.

[Hinton, 1981a] Geoffrey Hinton. *Implementing Semantic Networks in Parallel Hardware*, pages 161–188. Lawrence Erlbaum Associates, 1981.

[Hinton, 1981b] Geoffrey Hinton. Shape representation in parallel systems. In *Proceedings of the 7tyh International Joint Congress on Artificial Intelligence, B.C.*, pages 1088–1096, 1981.

[Hinton *et al.*, 1984] Geoffrey Hinton, Terrance Sejnowski, and David Ackley. *Boltzmann Machines: Constraint Satisfaction Networks that Learn.* Technical Report, Computer Science Department, Carnegie Mellon University, 1984.

[Hinton, 1987] Geoffrey E. Hinton. *Connectionist Learning Procedures.* Technical Report CMU-CS-87-115, Computer Science Department, Carnegie-Mellon University, 1987.

[Hinton *et al.*, 1986] G. E. Hinton, J. L. McClelland, and D. E. Rumelhart. Distributed representations. In David E. Rumelhart and James L McClelland, editors, *Parallel Distributed Processing: Exploring the Microstructure of Cognition, Volume 1: Foundations.*, pages 77–109, The MIT Press, Cambridge, Massachusetts, 1986.

[Holland *et al.*, 1986] John H. Holland, Keith J. Holyoak, Richard E. Nisbett, and Paul R. Thagard. *Induction: Processes of Inference, Learning and Discovery.* MIT Press, Cambridge, Massachusetts, 1986.

[Hopcroft and Ullman, 1979] J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley, Reading, Massachusetts, 1979.

[Jordan, 1985] M. I. Jordan. *Serial Order: A Parallel Distributed Processing Approach.* Technical Report 8604, Institute for Cognitive Science, University of California, San Diego, 1985.

[Kearns *et al.*, 1987] Michael Kearns, Ming Li, Leonard Pitt, and Leslie G. Valiant. Recent results on boolean concept learning. In *Proceedings of the Fourth International Workshop on Machine Learning*, pages 337–352, 1987.

[Kirkpatrick *et al.*, 1983] S. Kirkpatrick, Jr. C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

[Lippmann, 1987] Richard P. Lippmann. An introduction to computing with neural nets. *IEEE ASSP Magazine*, :4–22, April 1987.

[Lynne, 1988] Kenton Lynne. Competitive reinforcement learning. 1988. Submitted to 1988 IEEE Machine Learning Conference.

[McClelland, 1985] James L. McClelland. Putting knowledge in its place: A scheme for programming parallel processing structures on the fly. *Cognitive Science*, 9(1):113–146, 1985.

[McClelland and Kawamoto, 1986] J. L. McClelland and A. H. Kawamoto. Mechanisms of sentence processing: assigning roles to constituents of sentences. In D. E. Rumelhart and J. L. McClelland, editors, *Parallel Distributed Processing*, Bradford Books/MIT Press, 1986.

[McClelland and Rumelhart, 1986] J. L. McClelland and D. E. Rumelhart, editors. *Parallel Distributed Processing: Exploring the Microstructure of Cognition, Volume 2: Psychological and Biological Models.* Bradford Books/MIT Press, Cambridge, MA, 1986.

[Michalski *et al.*, 1983] Ryszard S. Michalski, Jaime G. Carbonell, and Tom M. Mitchell, editors. *Machine Learning.* Tioga Publishing Company, Palo Alto, California, 1983.

[Michalski *et al.*, 1986] Ryszard S. Michalski, Jaime G. Carbonell, and Tom M. Mitchell, editors. *Machine Learning : An Artificial Intelligence Approach, Volume II.* Morgan Kaufman Publishers, Inc., Palo Alto, CA, 1986.

[Minsky, 1985] Marvin Minsky. *The Society of Mind.* Simon and Schuster, New York, 1985.

[Minsky and Papert, 1969] Marvin Minsky and S. Papert. *Perceptrons.* MIT Press, Cambridge, Massachusetts, 1969.

[Mitchell, 1985] Tom Mitchell. Version spaces: A candidate elimination approach to rule learning. In *Proceedings of IJCAI 77*, pages 305–310, 1985.

[Parker, 1985] David B. Parker. *Learning Logic.* Technical Report, Center for Computational Research in Economics and Management Science, MIT, 1985.

[Plaut, 1986] David C. Plaut. *Experiments on Learning by Back Propagation.* Technical Report, Computer Science Department, Carnegie-Mellon University, 1986.

[Pollack, 1988] Jordan B. Pollack. *Recursive Auto-Associative Memory: Devising Compositioinal Distributed Representations.* Technical Report MCCS-88-124, Computing Research Laboratory, New Mexico State University, 1988.

[Rosenblatt, 1962] Frank Rosenblatt. *Principles of Neurodynamics.* Spartan, New York, 1962.

[Rumelhart and McClelland, 1986] D. E. Rumelhart and J. L. McClelland, editors. *Parallel Distributed Processing: Exploring the Microstructure of Cognition, Volume 1: Foundations.* Bradford Books/MIT Press, Cambridge, MA, 1986.

[Rumelhart and Zipser, 1985] David E. Rumelhart and David Zipser. Feature discovery by competitive learning. *Cognitive Science*, 9(1):75–112, 1985.

[Rumelhart *et al.*, 1986] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning internal representations by error propagation. In D. E. Rumelhart and J. L. McClelland, editors, *Parallel Distributed Processing: Exploring the Microstructure of Cognition, Volume 1: Foundations.*, Bradford Books/MIT Press, Cambridge, MA, 1986.

[Schlimmer, 1987] Jeffrey C. Schlimmer. Incremental adjustment of representations for learning. In *Proceedings of the Fourth International Workshop on Machine Learning*, pages 79–90, 1987.

[Schlimmer and Granger, 1986] Jeffrey C. Schlimmer and Richard H. Granger, Jr. Incremental learning from noisy data. *Machine Learning*, 1(3):317–354, 1986.

[Selman, 1985] Bart Selman. *Rule-Based Processing in a Connectionist System for Natural Language Understanding.* Technical Report CSRI-168, Computer Systems Research Institute, University of Toronto, 1985. (see also proceedings of CogSci 85).

[Selman and Hirst, 1985] Bart Selman and G. Hirst. A rule-based connectionist parsing system. In *Program of the Seventh Annual Conference of the Cognitive Science Society*, pages 212–221, Irvine, CA, 1985.

[Shastri, 1985] Lokendra Shastri. *Evidential Reasoning in Semantic Networks: A Formal Theory and its Parallel Implementation.* PhD thesis, University of Rochester, 1985.

[Shastri and Feldman, 1984] Lokendra Shastri and Jerome A. Feldman. *Semantic Networks and Neural Nets.* Technical Report 131, Computer Science Department, University of Rochester, 1984.

[Smith and Medin, 1981] Edward E. Smith and Douglas L. Medin. *Categories and Concepts.* Harvard University Press, Cambridge, Massachusetts, 1981.

[Smolensky, 1987] Paul Smolensky. *On Variable Binding and the Representation of Symbolic Structures in Connectionist Systems.* Technical Report CU-CS-355-87, Computer Science Department, University of Colorado, 1987.

[Sutton, 1986] Richard S. Sutton. Two problems with backpropagation and other steepest descent learning procedures for networks. In *Program of the Eighth Annual Conference of the Cognitive Science Society*, pages 823–831, 1986.

[Touretzky and Hinton, 1985] David S. Touretzky and Geoffrey E. Hinton. Symbols among the neurons: details of a connectionist inference architecture. In *Proceedings of the 8th International Joint Conference on Artificial Intelligence*, pages 238–243, 1985.

[Waltz and Pollack, 1985] David Waltz and Jordan Pollack. Massively parallel parsing: a strongly interactive model of natuaral language interpretation. *Cognitive Science*, 9(1):51–74, 1985.

[Waltz and Feldman, 1987] David L. Waltz and Jerome Feldman, editors. *Connectionist Models and their Implications.* Ablex Publishing Company, Norwood, New Jersey, 1987.

[Watrous and Shastri, 1987] Raymond L Watrous and Lokendra Shastri. Learning phonetic features using connectionist networks: an experiment in speech recognition. In *Proceedings of the IEEE First International Conference on Neural Networks*, pages IV381–IV388, 1987.